5-2016

A DJ Robot

Robert Hure
Trinity University, rhure1@trinity.edu

Josh King
Trinity University, jking1@trinity.edu

Mechatronics Design Report: A DJ Robot

Mechatronics
ENGR 4376
Instructor: Dr. Kevin Nickels

Robert Hure (*pledged)* Josh King (*pledged)*

TABLE OF CONTENTS

1. DESIGN SUMMARY

This report discusses the design of an audio entertainment device incorporating movement, lights, and motion in order to entertain the user, shown in Fig. 1. The user inserts a source of music through an AUX cord, which controls the color of the lights display based on the frequency of music. Through a control box tethered to the device, shown in Fig. 2, the user controls the amplitude of rotary motion the lights move before returning back to its original position and repeating, which is controlled by a stepper motor. The user also has control of which color represents the bass, mid, and treble frequency bands through the use of a single push button that cycles through all of the options. There are also three modes that the light can be in: a smooth response to the music, a strobe response to the music, and an off state with a simple fade sequence. These modes are selected through the use of a three-position-switch with the middle state being the off state. The kill switch allows for the lights to be quickly turned off in the event that the strobes become disorienting. While the lights are in either of the on states, the Arduino will start the off fade sequence if it detects that no music is being played for 5 seconds, then switches back to the previous mode as soon as music starts playing; this prevents the lights from being completely off for too long. The functional diagram for the device is shown in Figs. 3 & 4.

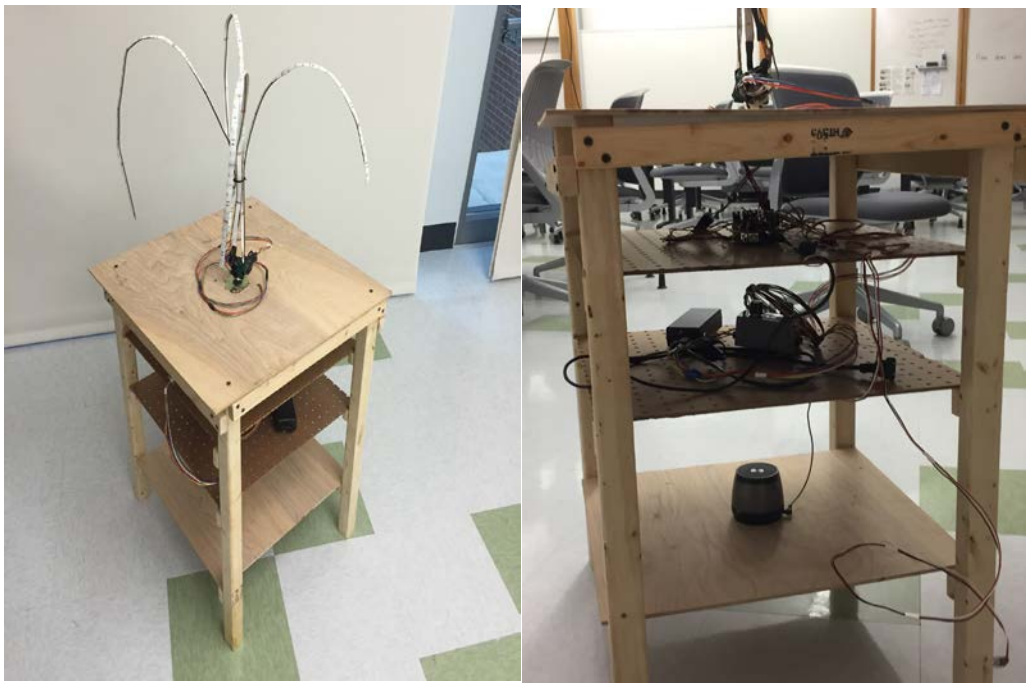


Figure 1. Overall design implemented

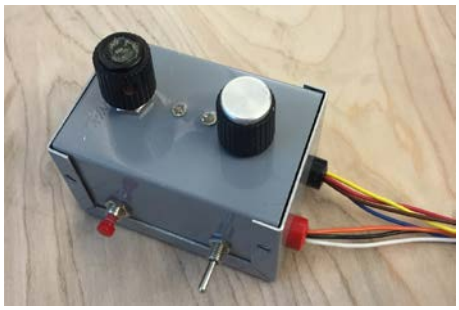


Figure 2. Control Box. Left knob controls motor sweep, right knob controls strobe time, button allows color selection, three position switch allows mode selection

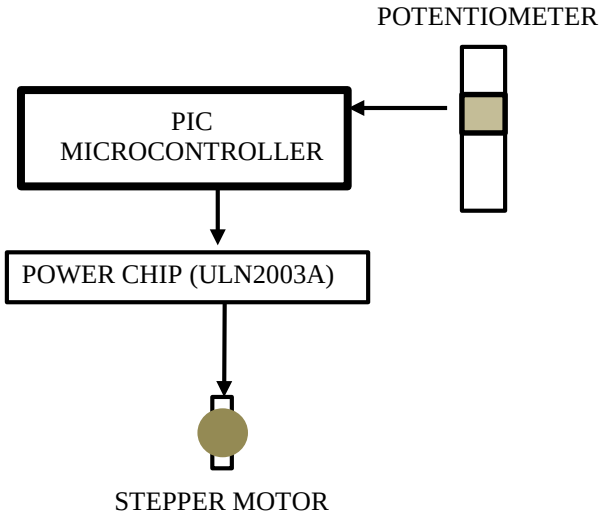


Figure 3. Functional Diagram for PIC

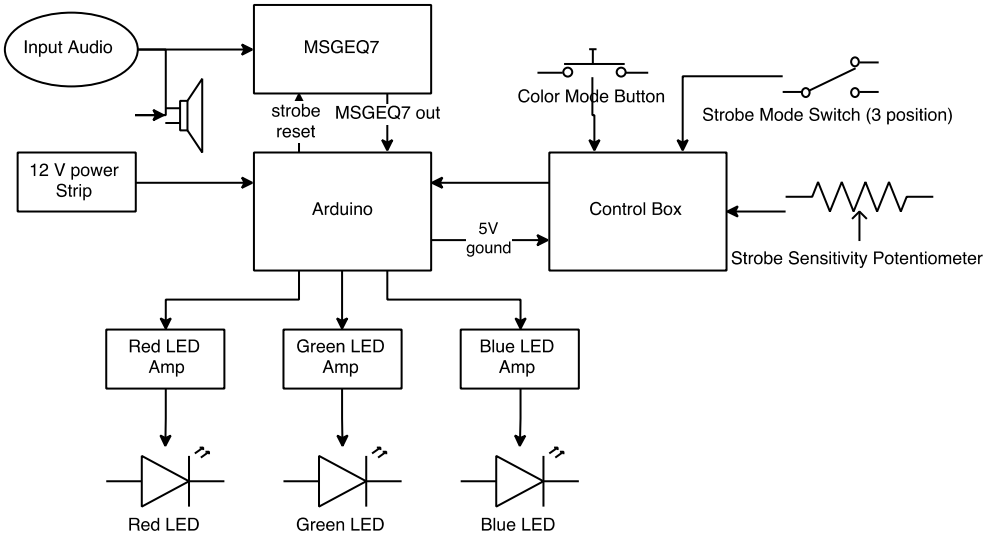


Figure 4. Function Diagram for Arduino

2. SYSTEM DETAILS

2.1 Output Display: Lights

Lights were attached to steel wire using the built-in adhesive on the back of the light strands. The light strands were coupled to the stepper motor by inserting wire into holes drilled into an attachment on the stepper motor, shown in Fig. 5. Lights were cut into segments for each piece of wire. One end of each light segment was exposed so that wires could be soldered to power and control the lights. The LED strips were powered by a 12V supply using the circuit shown in Fig. 6. The TIP 110 was chosen because it the only BJT in the lab rated to handle the 2A and 12V required by the light strip. The strips were sectioned into sets of three RGB LEDs where the strip could be cut across the copper contacts; cutting the strips in in other locations ruins that section of lighting.

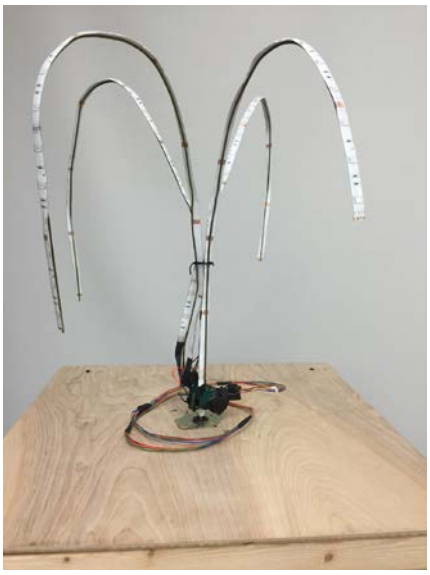


Figure 5. Light Display

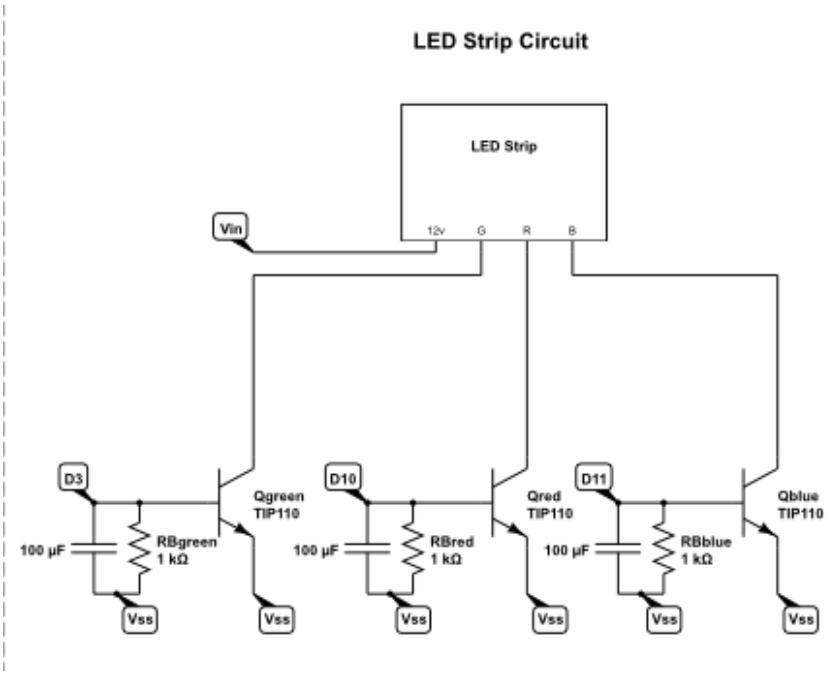


Figure 6. Circuit Schematic for the LED strips

2.2 Audio Output Device: Jam Speaker

The audio input from the user was split to send the same signal to the Arduino controlling the light system as well as the audio device, which was a small Jam personal speaker that can be purchased online for $25, shown in Fig. 7. The speaker runs on a battery with a charge of roughly 6 hours, according to the manufacturer. A home-made audio splitter was made for this device.

Figure 7. Jam Speaker used in design

2.3 Manual User Input: Control Box

The control box consists of a small metal casing containing two potentiometers, a NO push button, and a three-position switch. 5V and ground are supplied from the Arduino. The schematic for the control box is shown in Fig. 8. The Arduino reads the strobe potentiometer while the motor pot is read by the PIC. The switch connects the D6 pin to 5V when it is toggled up, neither pin is connected to 5V when the switch is toggled down, and the D7 pin is connected to 5V when the switch is toggled down. The Arduino interoperates the values of these two pins to determine the state of the switch and set the mode of the lights. When the push button is pressed, 0V is applied to the D4 pin of the Arduino, causing cycling through the color combinations.

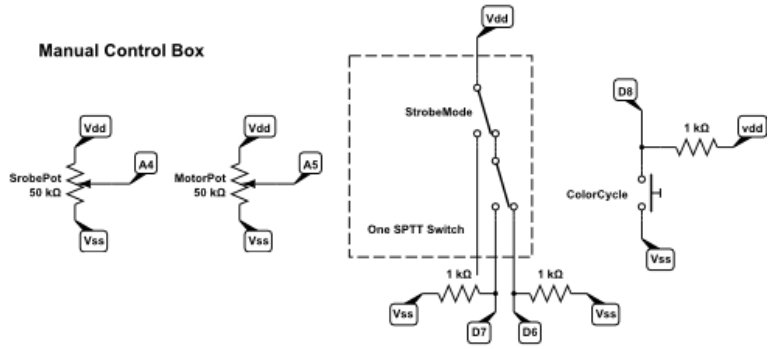


Figure 8. Schematic for manual control box

2.4 Automatic Input: Music

The audio signal acts as an automatic input into this system. It is interpreted by the MSGEQ7 music chip in the schematic shown in Fig. 9. This chip provides analog readings for 7 different frequency bands. The Arduino sends a pulse to the strobe pin to signal the IC to send the next value. Once it has received all seven values it signals the chip to reset. The values received control the brightness of RGB LEDs, causing the color assigned to the particular band to increase in brightness when a higher reading is being received for that band. The lights will also switch to the fade sequence when no music being received.

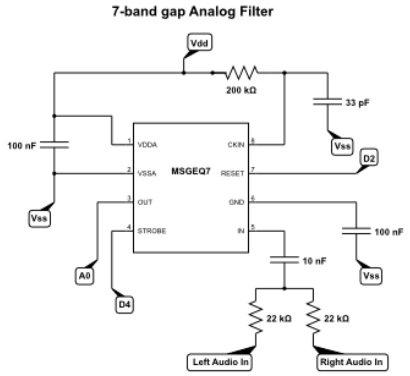


Figure 9. Schematic for MSGEQ7 music chip

2.5 Hardware: Stepper Motor

A stepper motor was used to spin the light display, shown in Fig. 5. Signals sent from the PIC are amplified to 12 V by the ULN2003 and sent to the stepper motor to energize different coils in sequence to rotate a shaft, shown in Fig. 10. The stepper motor schematic can be seen in Fig. 11. The size of the motor and the load attached to the stepper motor determine how fast the motor can spin. If the step time is too small or the load is too great, the stepper will not progress from step to step and may stall or process backwards with a great enough load imbalance. To combat this, a brief transition phase was added between steps. Similar to microstepping, the previous coil and the next coil are energized in the transition phase, preparing the stepper to transition into the next stage. By attaching a load and observing minimum step time before failure, it was determined that addition of the transition phase increased motor torque. Microstepping was not implemented for simplicity, as microstepping requires four PWM outputs.

Once the load was attached, the minimum step time was tuned and found to be 0.6 s/step. An analog input from a potentiometer in the control box determined how many steps the motor would take in the clockwise direction before reversing and taking that many steps in the counterclockwise direction to return to the original position and take another reading from the potentiometer.

Stepper motors are well suited for position control applications, which was important to the design; wiring to lights was connected to the main control circuitry by a tether of wires. An uneven amount of spinning by the light display would put the tether under tension, damaging the control circuitry and light display. Stepper motors are also quieter than PMDC motors, which is important in audio applications.

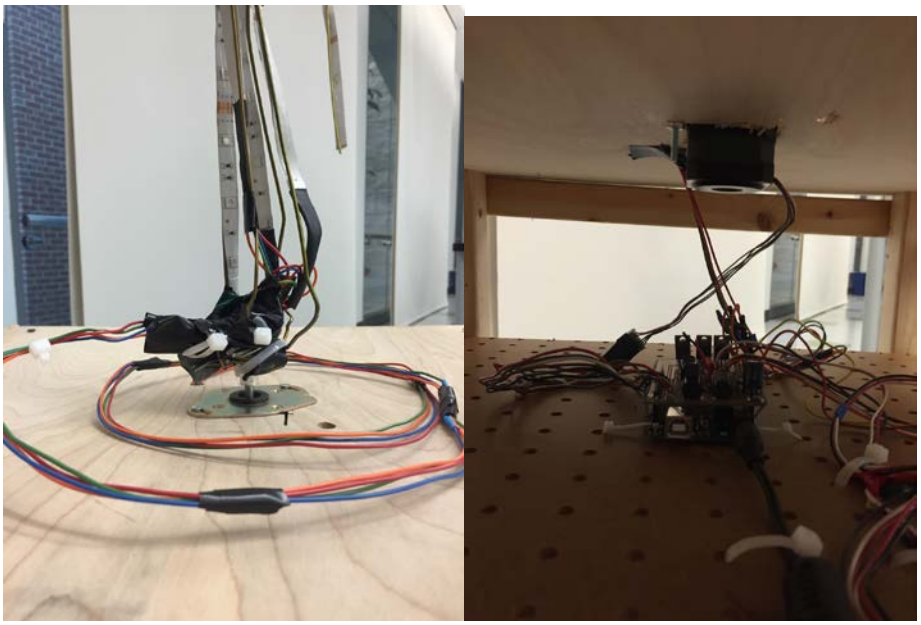


Figure 10. Stepper motor mounted to wooden frame from top (left) and bottom (right)

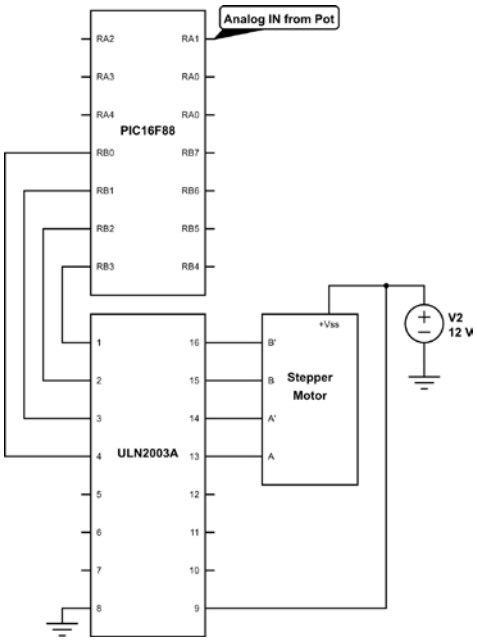


Figure 11. Stepper motor schematic

2.6 Logic & Processing: Arduino & PIC

A Microchip PIC16F88 microcontroller was used to read an analog input from a potentiometer and send out four digital signals to control the stepper motor. The code implemented follows the flowcharts shown in Figs. 12 & 13. The entire code can be seen in the Appendix. The PIC reads the input from the potentiometer through the ADC and uses this value

to set the amplitude of the stepper motor sweep, which has a maximum of roughly 1.5 rotations and a minimum of no rotation. The stepper motor then moves clockwise an amount equal to the amplitude, then moves back.

The Arduino adjusts the brightness of the red, green, and blue LEDs to achieve a color that represents the amplitude of each frequency band. To do this the Arduino preforms a statistical analysis of the three major frequencies bands (bands 1, 3, and 5) using a downloaded statistics library. This library can return the average and standard deviation of a collected data set. The first thing the Arduino does for each run is it checks the size of each statistics data set and resets them if they reach the maximum. When the data set is reset the last two readings are added to the new data set to prevent any errors that might be cause by statistical analysis on an empty data set.

After the Arduino confirms the sample sizes of the statistics, it reads the input values from the control box. The first component checked is the push button, which the Arduino will only respond to if it is both low and not equal to its state from the previous run. This prevents multiple reads being registered from one press. When it is registered that the button is pressed, the Arduino increments the count for the color mode and tells the rest of the code that a new color mode has been choses and to reset to the initial condition for the zero mode through Boolean variables. After this the code stores the state of the pushbutton to compare with the next run through and takes the readings of the two pins attached to the three-position switch. If it detected that the strobe on pin is high it tells the code that it is in the strobe mode and that it is not in the kill mode, if it detects that the strobe off pin is high it tells the code that it is in strobe off mode and that it is not in kill mode, and if it detects that both pins are low it tell the code that it is in kill mode.

Finally it will read and map the value of the strobe sensitivity potentiometer. After it has read the inputs from the control box, the Arduino checks to see if it needs to change the color mode. If the Boolean variable tells the Arduino that it does not need to change the color mode the code advances to the next step. However, if the code does need to change the color mode it will set the Boolean variables for each color and frequency combination based on the value generated from the control box reading and then tells the code that it no longer needs to change the color mode.

Next the Arduino takes the readings from the MSGEQ7 chip after sending it the command to reset. This is done by sending a low value to the strobe pin on the chip and taking and ADC input from the chip. Once the reading is taken, the value is constrained to remove noise, mapped to the same range that the LEDs can receive, and the difference from the previous reading is calculated. This is repeated for all frequency ranges using a for loop. Once all the values have been collected, the three major bands are added to their respective statistics data set. The next step is to determine if all of the readings were zero. When all bands are completely off, the code resets the nonzero count and increments the zero count. When at least on band is above 0, the code increments the nonzero count and if it reaches 5 then the code resets the zero count. This removes the possibility of the code registering small levels of interference as a music input.

Now that all the necessary inputs have been taken the code can decide on which state it is supposed to be in. If the code is told to enter the zero state, it first checks to see if it has also been told to reset in zero mode. If it has been told to reset then it sets the bass value to the maximum and the middle and treble values to 0. Otherwise the code will run through a slow fade sequence that can be sped up or slowed down by changing the delay constant in the code. The colors on this fade sequence are also altered using the pushbutton.

The fade sequence preforms one step and then returns to the beginning of the main loop allowing for the lights to exit the zero mode almost instantly if music is detected. If the code is not told to enter the zero mode, it will run through the process of defining the minimum, maximum, step size, and finally value for each light.

The maximum of each light is decided on which band has the greatest amplitude with the largest reading having a maximum of 255, the second highest having a maximum of 200, and the lowest having a maximum of 150. This idea behind this limitation of the colors is to help prevent the bright white light that can be cause if the input volume is too high and to also ensure that more unique colors are provided. The minimum value of each light is set the reading from the MSGEQ7 preventing the light from going of if a steady value is being received. If the MSGEQ7 reading is larger than the maximum, then the minimum is set to the maximum. The step size for each light is defined by comparing the current reading to the running average. For a reading larger than 1 standard deviation away from the average the step size is set to twice the upward step size. For a reading between the average and 1 standard deviation above the average, the step size is set to the upward step size. For a reading between the average and 1 standard

deviation below the average, the step size is set to the downward step size. For a reading 1 standard deviation below the average, the step size is set to twice the downward step size. The section of the code that define the values for each light will check to see if the step size puts the value above or below the minimum or maximum and sets it the respective value if it does. Otherwise, the code will set the value to the previous value plus the decided step size.

Now that every value has been defined, the Arduino will either implement the strobe on conditions to the lights or the strobe off conditions. In the off strobe mode, the code will adjust the Boolean variables for the color and band combinations and set the values for each LED accordingly. The strobe on mode works the same way as the strobe on mode, but it will limit the values of the bands based on the difference readings in relation to the strobe sensitivity. This allows for rapid changes in the amplitude of the frequency bands to be displayed on the light strands. The lights respond better to the music in this mode, but, since the strobes can be disorienting to people, the option to turn it off and adjust the sensitivity is given. At this point the code return to the beginning of the main loop keeping all the values recoded from the last run as only global variable are used. The flow charts and code for the Arduino processing can be found in the Appendix.

The entire circuit was constructed on a printed circuit board that connected into the Arduino, similar to a test shield, as shown in Fig. 14. The printed circuit board is two sided based on the number of components needed and is roughly 2.5” x 4”. The overall schematic is shown in the Appendix.

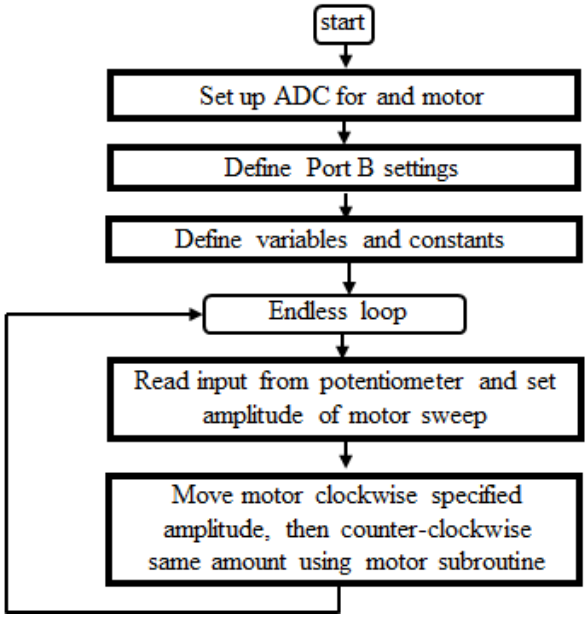


Figure 12. Flowchart followed by PIC to control stepper motor

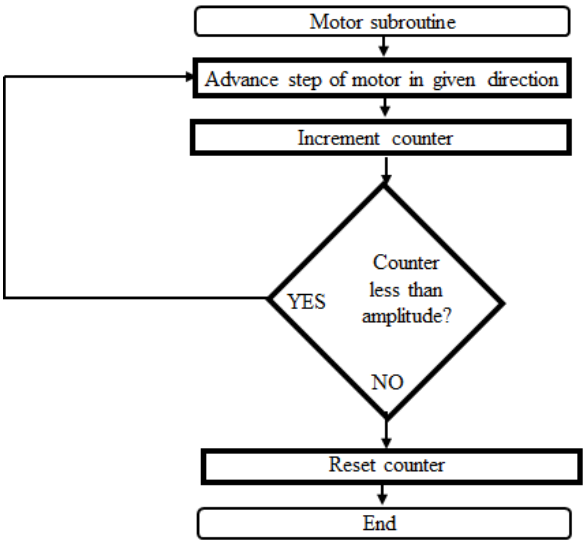


Figure 13. Flowchart of subroutine to move stepper motor

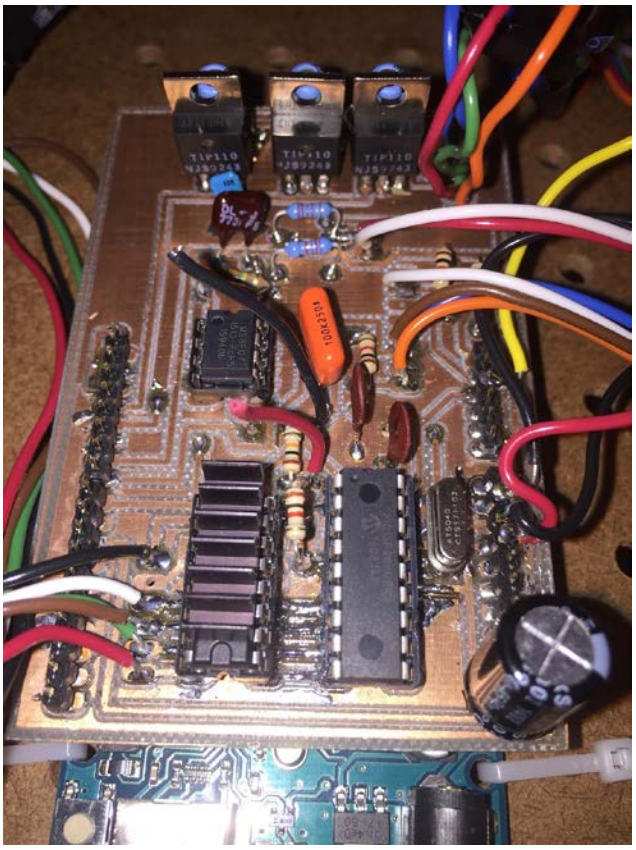


Figure 14. Printed Circuit Board atop Arduino with all components soldered on. Foreground from the right: 470 uF capacitor to level the power supply, PIC16F88, ULN2003A. Middleground: Audio processing chip. Background: TIP110 BJTs to power lights.

3. DESIGN EVALUATION

The implemented design functioned in all listed design categories. The light strands changed color based on Arduino inputs to span the entire RGB range. The lights responded the music quickly and exactly as desired. Every type of music showed up well in either the strobe or fade setting. The transition to and from the off state were smooth and quick. The Jam Speaker attached to the home-made audio splitter produced clear sound at appropriate volumes. The control box allowed the user to alter the light display and rotation of the lights using two potentiometers and a button.

The stepper motor successfully rotated the light system around and accepted inputs from the potentiometer. Difficulties were experienced with the stepper motor; some back-rotation did occur due to the size of the load, which became a greater problem with use. Understanding how to interface with the stepper motor and increase torque required extensive research from the assigned textbook, internet sources, Trinity University's Electronics Shop, and the instructor.

Implementing the transition phase to increase torque as a simpler version of microstepping was the author's original idea after reading about microstepping.

The code was able to process and adjust the lights faster than the human eye can detect. Unique colors were created from different songs and the lights were successfully prevented from all being at max brightness. There was no lag time switching between color modes and strobe states

The project proposal deliverable received a grading adjustment of +2, and our device was demonstrated on the early early-bird date for a grading adjustment of +10. All components were well integrated on a tidy printed circuit board, and all connections were soldered. Connecting wires were taped together and tied down for neatness. The Arduino-PCB combination was zip-tied down to the device for durability, shown in Fig. 15. Creating the PCB required extensive work on the author's part and the shop technician's part. The authors learned how to use the software Eagle to design a printed circuit board, then re-designed the circuit board twice after receiving feedback from the shop technicians. The authors' soldering skills improved significantly from creating the PCB.

The speaker ran on a battery supply. All other components were powered from one wall plug connected to an AC/DC converter to deliver 12 V to the stepper and lights, which was brought down to 5 V by the Arduino for all other components.

A PIC was used to control the stepper motor, while an Arduino was used to control the lights. The Arduino was able to handle the load placed on it, but the 12V power source did cause it to heat up a fair amount.

Initially the design implemented 4 PICs to control the lights instead of the Arduino. One PIC would read the values from the MSGEQ7 and send them to the three other PICs, which would interoperate them and send a PWM command to their respective light color. The reason PICs were not used for light control was due to the difficulties that arose when attempting to get the PICs to communicate with each other in slave and master modes. The use of an Arduino also allowed for the implementation of the statistics library, which would have been much more difficult with the PICs.

Figure 15. Implemented wiring. All connections are soldered, all wires are zip tied down, all wires going to the same location are taped together

4. PARTIAL PARTS LIST

Table 1. Partial Parts List

Component/Description	Part Number	Vendor	Price
Stepper Motor	4017-868	Trinity Electronics Shop	~$40
Current Amplifier for Stepper	ULN2003A	Trinity Electronics Shop	$0.50
Speaker	Jam Personal Speaker	Best Buy	$30
Microcontroller	PIC16F88	Trinity Electronics Shop	$1
Arduino	Arduino Uno	Trinity Electronics Shop	$25
Graphic equalizer chip used to process the audio signal	MSGEQ7	Adafruit	$3
3X BJTs used as amplifiers for the LED strip.	TIP 110	Trinity Electronics Shop	$1
LED light strip	TaoTronics LED light Strip	Amazon	$20

5. LESSONS LEARNED

We faced difficulties implementing the printed circuit board (PCB). Based on the number of components and size restraints to fit atop the Arduino Uno, a two-sided circuit board was required. Our PCB was the first two-sided board made at Trinity University, which required additional research by Trinity's shop technicians. Over a full week, roughly 12 hours were spent

by the author designing the circuit board, 4 hours by the shop technician preparing the board for printing, and 4 hours for the printer to create the board.

After the board was printed, it was discovered that each component had to be soldered on both sides of the PCB, doubling the number of required solders. Soldering on both sides of the PCB proved particularly difficult for sockets holding the PIC, ULN, and music chip, as solder had to be slid between the plastic socket and the PCB in a gap the width of solder wire. Two traces on the PCB were severed during the soldering process and had to be reattached with solder. Roughly 15 hours were spent soldering on the PCB. It is recommended that continuity is checked between all pins.

There was difficulty sizing the stepper motor. In order to ensure proper rotation of the lights, the length of lights and speed of rotation had to be reduced. If the authors were to redesign the system, a more powerful motor would be used to allow faster spinning of more lights, which would be more entertaining.

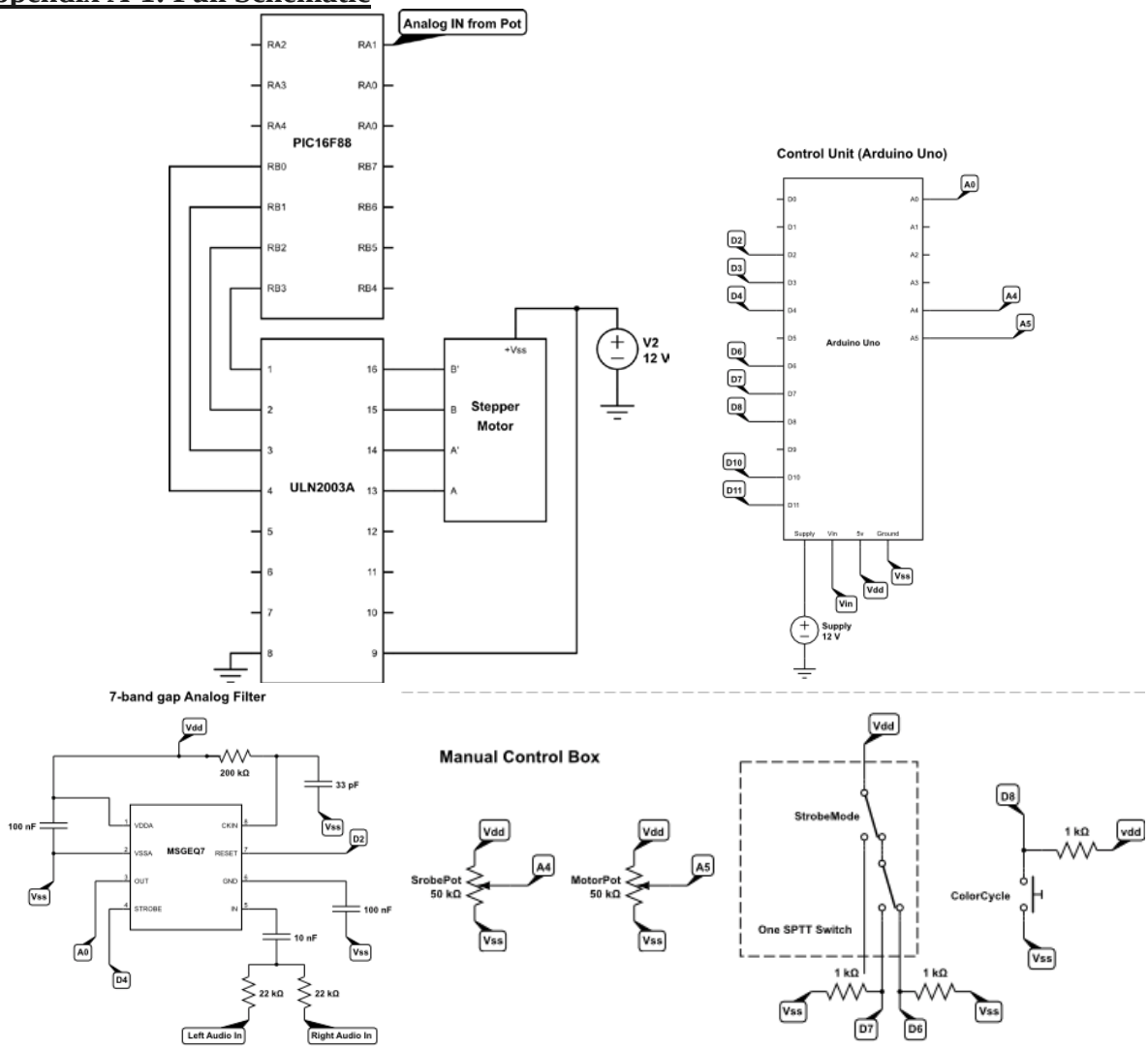
The lights had trouble with clipping if the music input was placed much above half values. The solution to this implemented in this design was to just have the user turn down the volume on the input device. However, this limits how loud the user can play the music out of the speaker. A solution that was discussed but not implemented due to time was to add a separate amplifier for the music signal going into the MSGEQ7 to prevent clipping without affecting the sound of the music coming out of the speaker.

6. ACKNOWLEDGEMENTS

The authors would like to acknowledge Trinity University Shop Technicians Ernest Romero and Marc Trestman for their advice and help finding components, selecting components, and creating the printed circuit board. The authors would like to acknowledge the instructor, Dr. Kevin Nickels, for his advice throughout the project and for guiding research on controlling the stepper motor.

7. APPENDIX

Appendix A-1: Full Schematic



Appendix A-2: PIC Code

```
'*****************************************************************
'*  Name    : Mechatronics Project General Code                  *
'*  Author  : Robert Hure                                        *
'*  Date    : 4/18/2016                                          *
'*  Notes   :  Mechatronics Project: DJ Robot                    *
'*****************************************************************

' Define ADCIN parameters
DEFINE  ADC_BITS        10      ' Set number of bits in result
DEFINE  ADC_CLOCK       3       ' Set clock source (3=rc)
DEFINE  ADC_SAMPLEUS    15      ' Set sampling time in uS

'Photoresistor ADC
adcVar2  VAR WORD    ' ADC  result
dutyCycle2 var byte  ' Duty cycle for PWM
```

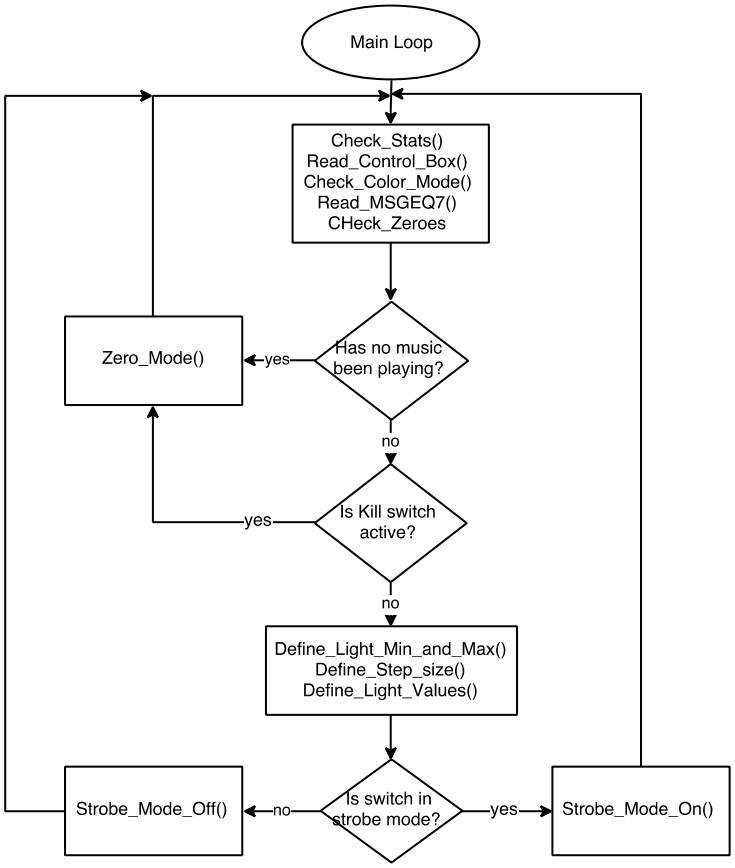
```
input porta.0
output porta.2

'Motor ADC
adcVar  VAR WORD    ' ADC  result
amplitude var byte  ' Duty cycle for PWM
input porta.1

' Set up ADCON1
ADCON1 = %10000000 ' Right-justify results (lowest 10 bits)

'Enable PORTB pull-ups
OPTION_REG = $7f

'Initialize I/O
TRISB = %00000000

'Define Variables
motor var Byte  'Used to store the step of the motor
i var byte      'Used for step counting
num_steps var byte  'Used to assign number of steps to move
motor_dir var byte   'Used to store direction motor is to rotate

'Define Constants
CW con 0
CCW Con 1
stage1 con %0001    'Stages stepper motor moves through during motion, corresponding to active coil
stage2 con %0010
stage3 con %0100
stage4 con %1000
step_delay con 400   'Amount of time spent in each stage (ms)
partstep con 15     'Amount of time spent between stages (ms)

motor = stage4      'Initial motor state assignment

'Main subroutine
move:
  while (1)     'Always loop
  ADCIN 1, adcVar    'Input to stepper motor
  ' convert ADC value to a byte value:
  amplitude = 10+ adcVar / 8 'Amplitude of sweep
  motor_dir = CW    'Declare direction and steps
  num_steps=amplitude
  gosub move_steps    'Move specified steps, then reverse
  motor_dir = CCW
  gosub move_steps
  wend          'loop forever

Return

'Subroutine to move the motor a given number of steps
move_steps:
  For i=1 to num_steps
     Gosub step_motor
     gosub lights    'check photoresistor
  Next i
```

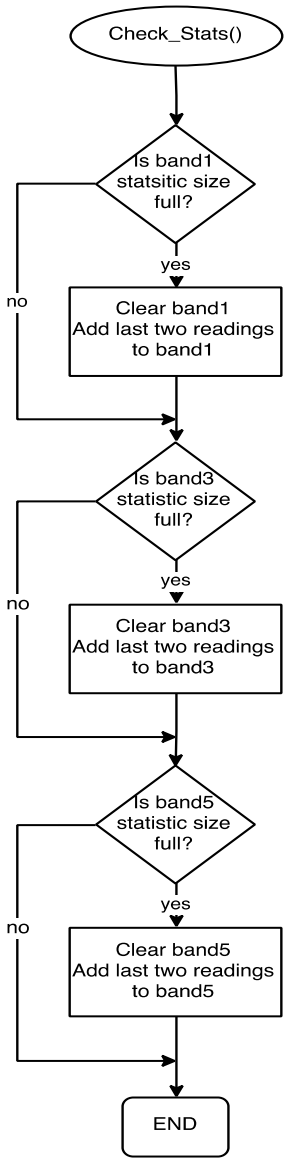
```
Return

'Subroutine to read photoresistor
lights:
   ADCIN 0, adcVar2
   dutyCycle2 = adcVar2 / 4
   if (dutyCycle2 > 100) then
   high porta.2
   else
   low porta.2
   endif
return

'Subroutine to move to the next step in the desired direction
step_motor:

  if (motor=stage1) and (motor_dir=CW)then
  motor=stage2       'Set the new motor stage
  gosub stage12      'Transition phase
  gosub stage2o      'Next motor phase
    else
   if (motor=stage2) and (motor_dir=CW) then
   motor=stage3
   gosub stage23
   gosub stage3o
   else
      if (motor=stage3) and (motor_dir=CW) then
      motor=stage4
      gosub stage34
      gosub stage4o

      else
         if (motor=stage4) and (motor_dir=CW)then
         motor=stage1
         gosub stage14
         gosub stage1o


         else
            if (motor=stage4) and (motor_dir=CCW)then
            motor=stage3
            gosub stage34
            gosub stage3o


            else
               if (motor=stage3) and (motor_dir=CCW)then
               motor=stage2
               gosub stage23
               gosub stage2o


               else
                  if (motor=stage2) and (motor_dir=CCW)then
                  motor=stage1
                  gosub stage12
```

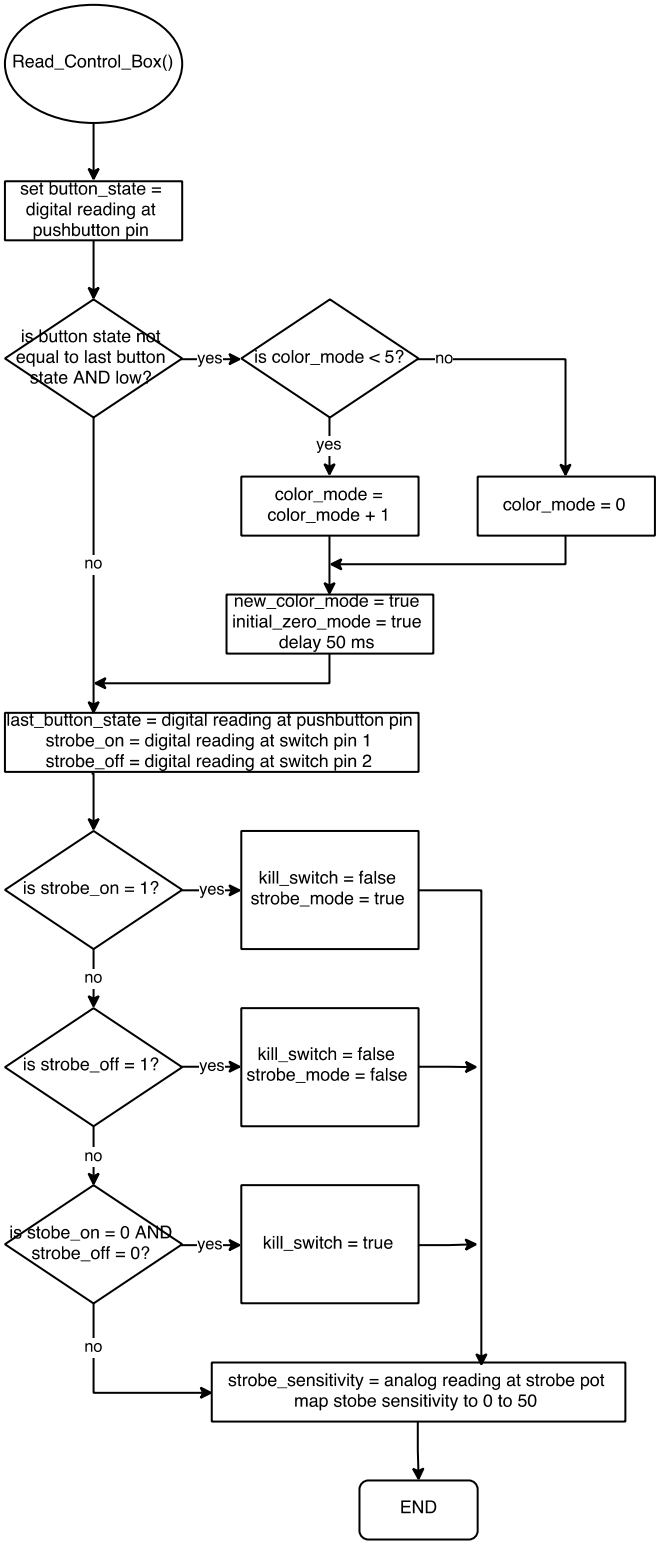
```
                gosub stage1o


                else
                   'if (motor=stage1) and (motor_dir=CCW)then
                   motor=stage4
                   gosub stage14
                   gosub stage4o

                   'endif
                endif
             endif
          endif
       endif
    endif
  endif
  pause step_delay
return

'Subroutines to command the motor
stage1o:
low portb.3       'Pin assignments for this stage
low portb.2
low portb.1
high portb.0
return

stage12:
low portb.3        'Pin assignments for transition stage
low portb.2
high portb.1
high portb.0
pause partstep
return

stage2o:
low portb.3
low portb.2
high portb.1
low portb.0
return

stage23:
low portb.3
high portb.2
high portb.1
low portb.0
pause partstep
return

stage3o:
low portb.3
high portb.2
low portb.1
low portb.0
```

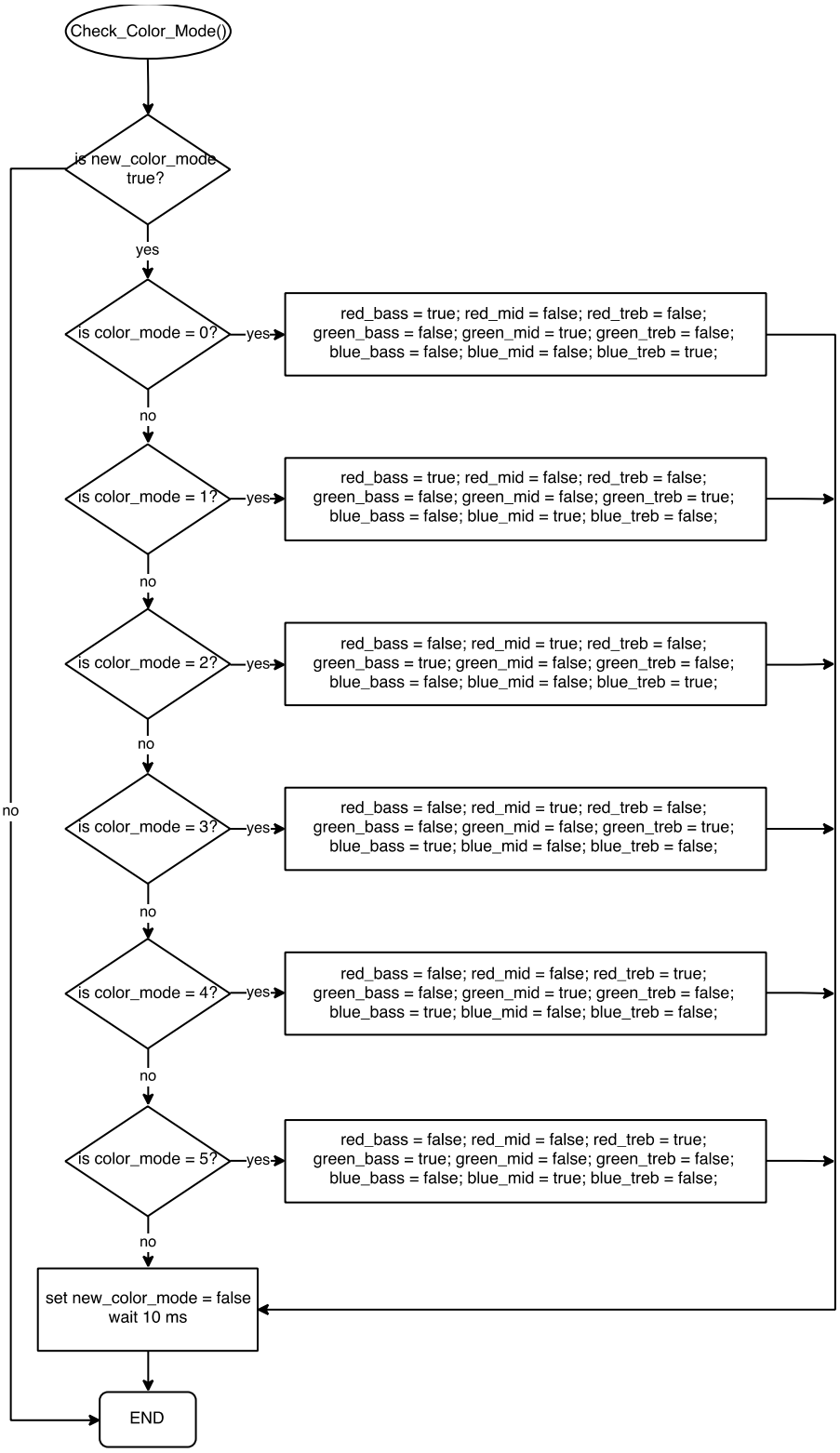
```
return

stage34:
high portb.3
high portb.2
low portb.1
low portb.0
pause partstep
return

stage4o:
high portb.3
low portb.2
low portb.1
low portb.0
return

stage14:
high portb.3
low portb.2
low portb.1
high portb.0
pause partstep
return
```

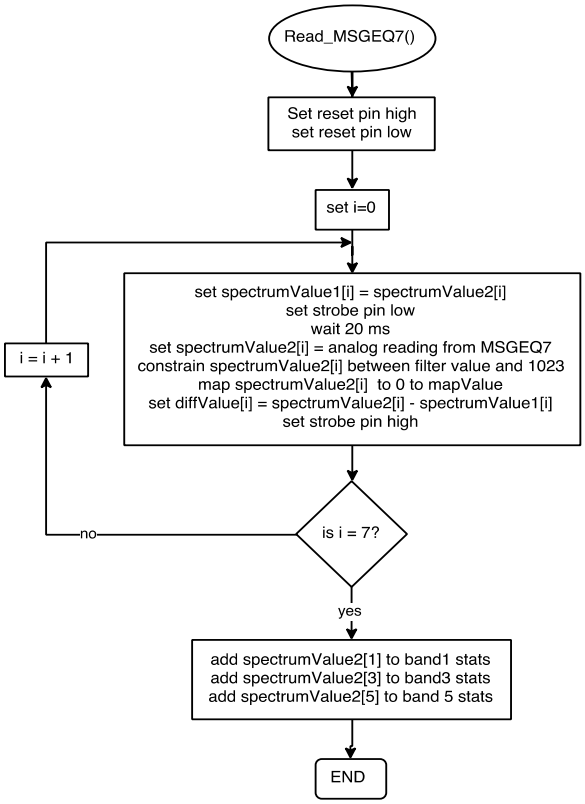
Appendix A-3: Arduino Flowcharts

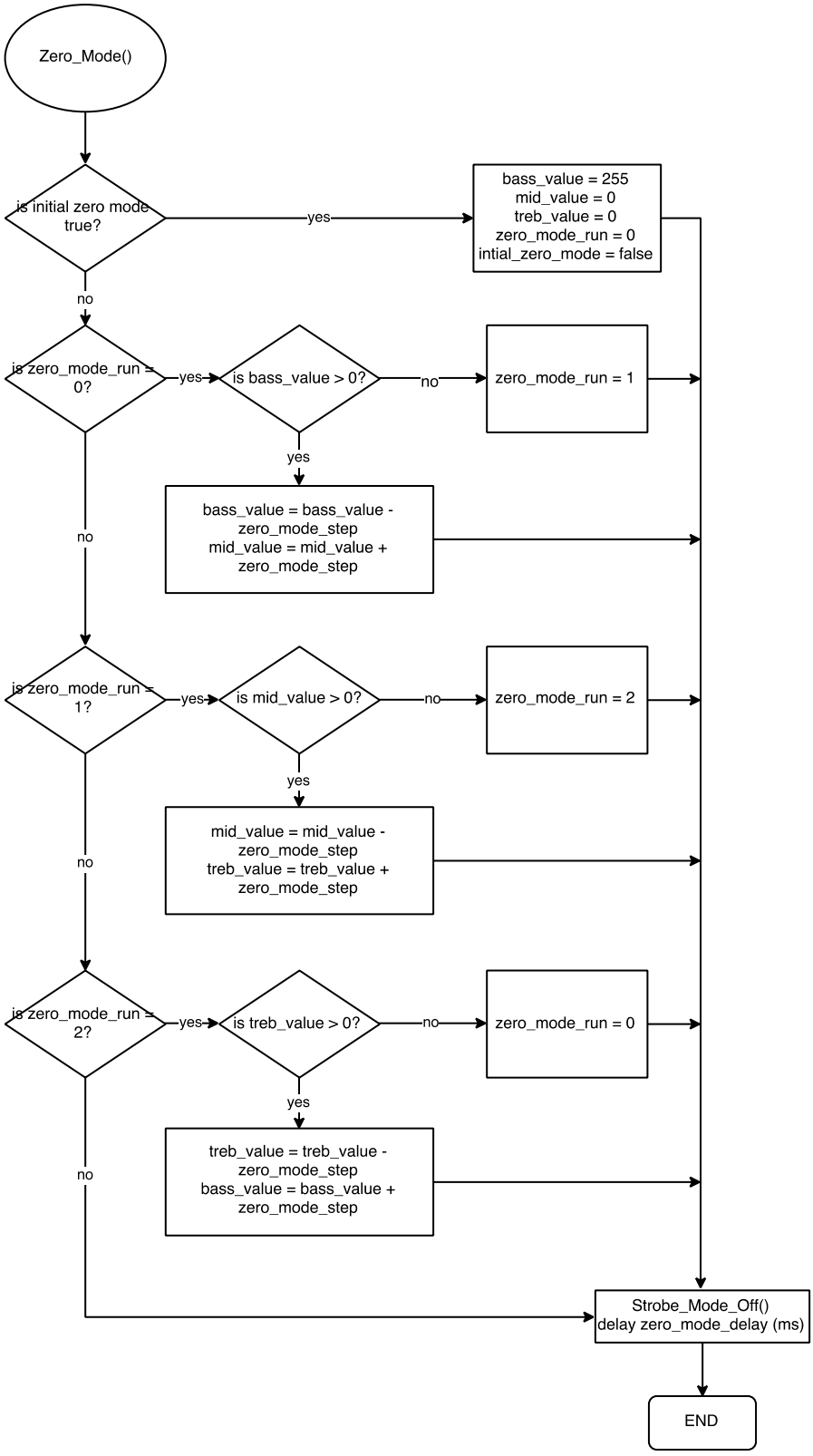


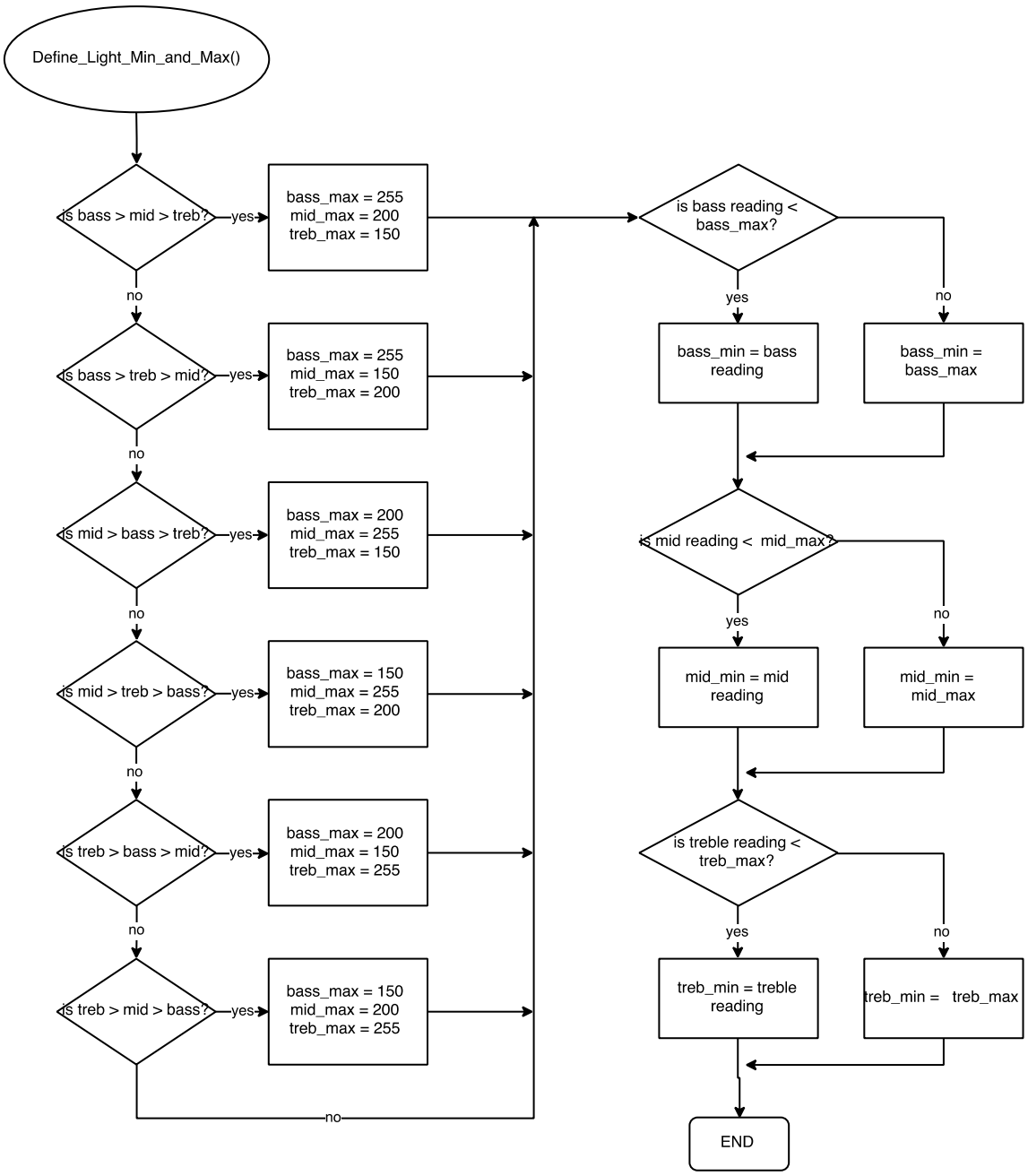

Check_Stats()
Is band1 statsitic size full?
yes
no
Clear band1
Add last two readings to band1
Is band3 statistic size full?
yes
no
Clear band3
Add last two readings to band3
Is band5 statistic size full?
yes
no
Clear band5
Add last two readings to band5
END

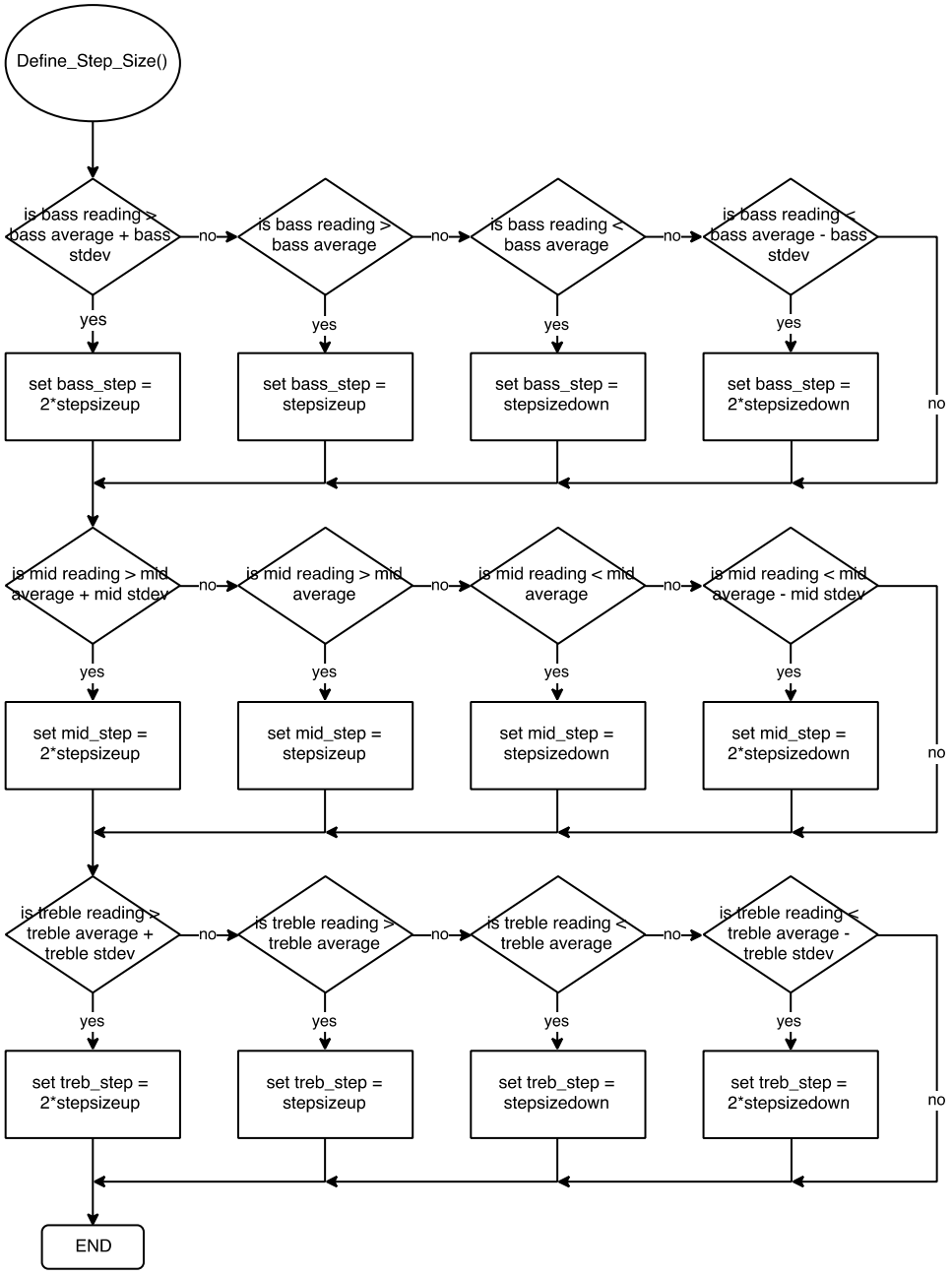

Read_Control_Box()
set button_state = digital reading at pushbutton pin
is button state not equal to last button state AND low?
yes
is color_mode < 5?
no
yes
color_mode = color_mode + 1
color_mode = 0
no
new_color_mode = true
initial_zero_mode = true
delay 50 ms
last_button_state = digital reading at pushbutton pin
strobe_on = digital reading at switch pin 1
strobe_off = digital reading at switch pin 2
is strobe_on = 1?
yes
kill_switch = false
strobe_mode = true
no
is strobe_off = 1?
yes
kill_switch = false
strobe_mode = false
no
is stobe_on = 0 AND strobe_off = 0?
yes
kill_switch = true
no
strobe_sensitivity = analog reading at strobe pot
map stobe sensitivity to 0 to 50
END

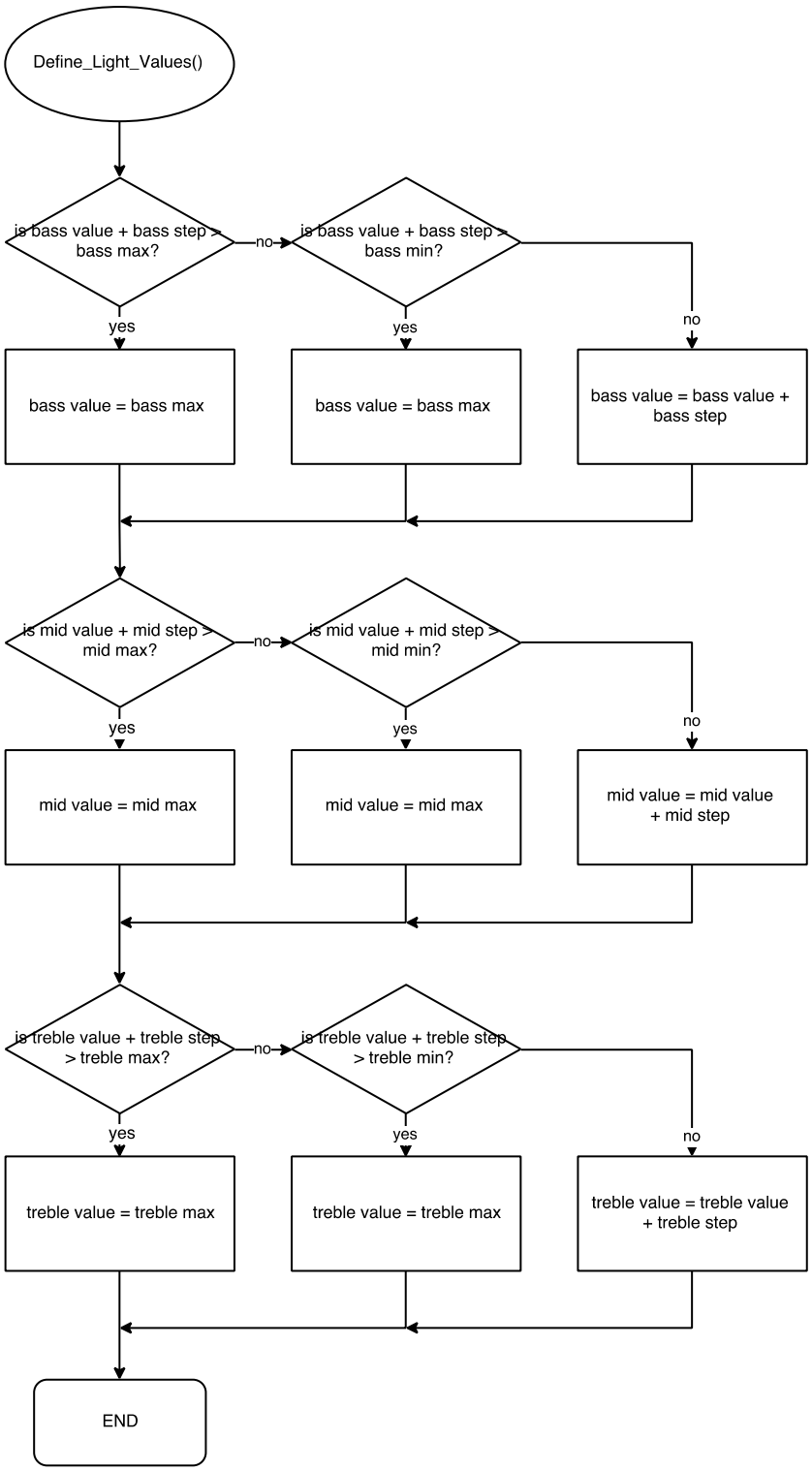

Check_Color_Mode()
is new_color_mode true?
yes
no
is color_mode = 0?
yes
red_bass = true; red_mid = false; red_treb = false;
green_bass = false; green_mid = true; green_treb = false;
blue_bass = false; blue_mid = false; blue_treb = true;
no
is color_mode = 1?
yes
red_bass = true; red_mid = false; red_treb = false;
green_bass = false; green_mid = false; green_treb = true;
blue_bass = false; blue_mid = true; blue_treb = false;
no
is color_mode = 2?
yes
red_bass = false; red_mid = true; red_treb = false;
green_bass = true; green_mid = false; green_treb = false;
blue_bass = false; blue_mid = false; blue_treb = true;
no
is color_mode = 3?
yes
red_bass = false; red_mid = true; red_treb = false;
green_bass = false; green_mid = false; green_treb = true;
blue_bass = true; blue_mid = false; blue_treb = false;
no
is color_mode = 4?
yes
red_bass = false; red_mid = false; red_treb = true;
green_bass = false; green_mid = true; green_treb = false;
blue_bass = true; blue_mid = false; blue_treb = false;
no
is color_mode = 5?
yes
red_bass = false; red_mid = false; red_treb = true;
green_bass = true; green_mid = false; green_treb = false;
blue_bass = false; blue_mid = true; blue_treb = false;
no
set new_color_mode = false
wait 10 ms
END

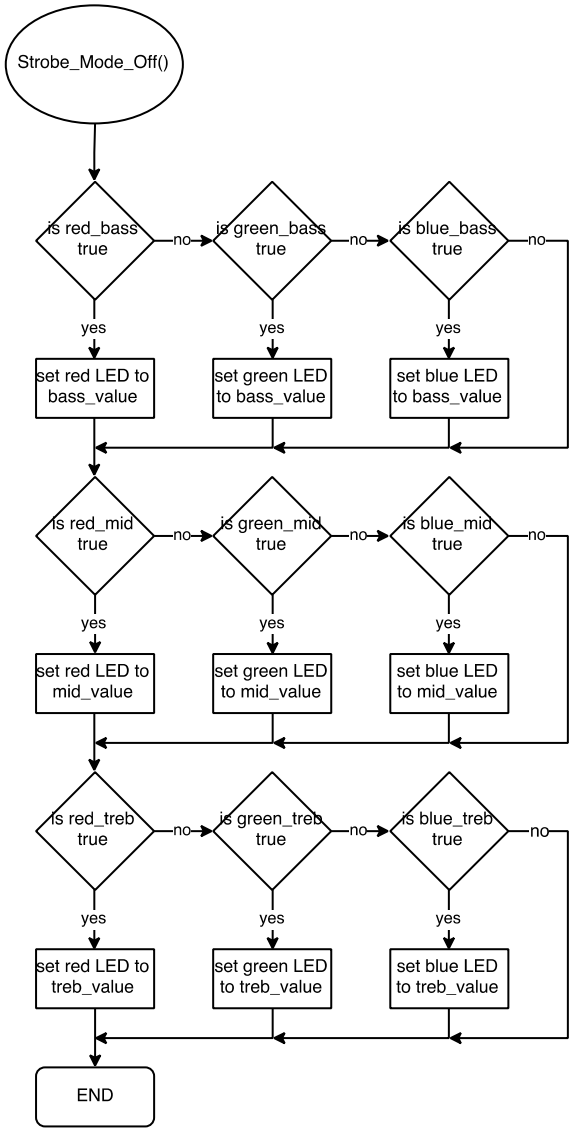

Read_MSGEQ7()
Set reset pin high
set reset pin low
set i=0
set spectrumValue1[i] = spectrumValue2[i]
set strobe pin low
wait 20 ms
set spectrumValue2[i] = analog reading from MSGEQ7
constrain spectrumValue2[i] between filter value and 1023
map spectrumValue2[i] to 0 to mapValue
set diffValue[i] = spectrumValue2[i] - spectrumValue1[i]
set strobe pin high
i = i + 1
is i = 7?
no
yes
add spectrumValue2[1] to band1 stats
add spectrumValue2[3] to band3 stats
add spectrumValue2[5] to band 5 stats
END
Check_Zeros()
are all bands 0?
yes
nonzerocount = 0
zerocount =
zerocount + 1
no
nonzerocount =
nonzerocount + 1
is nonerocount >
nonzerocountmax?
yes
zerocount = 0
no
END

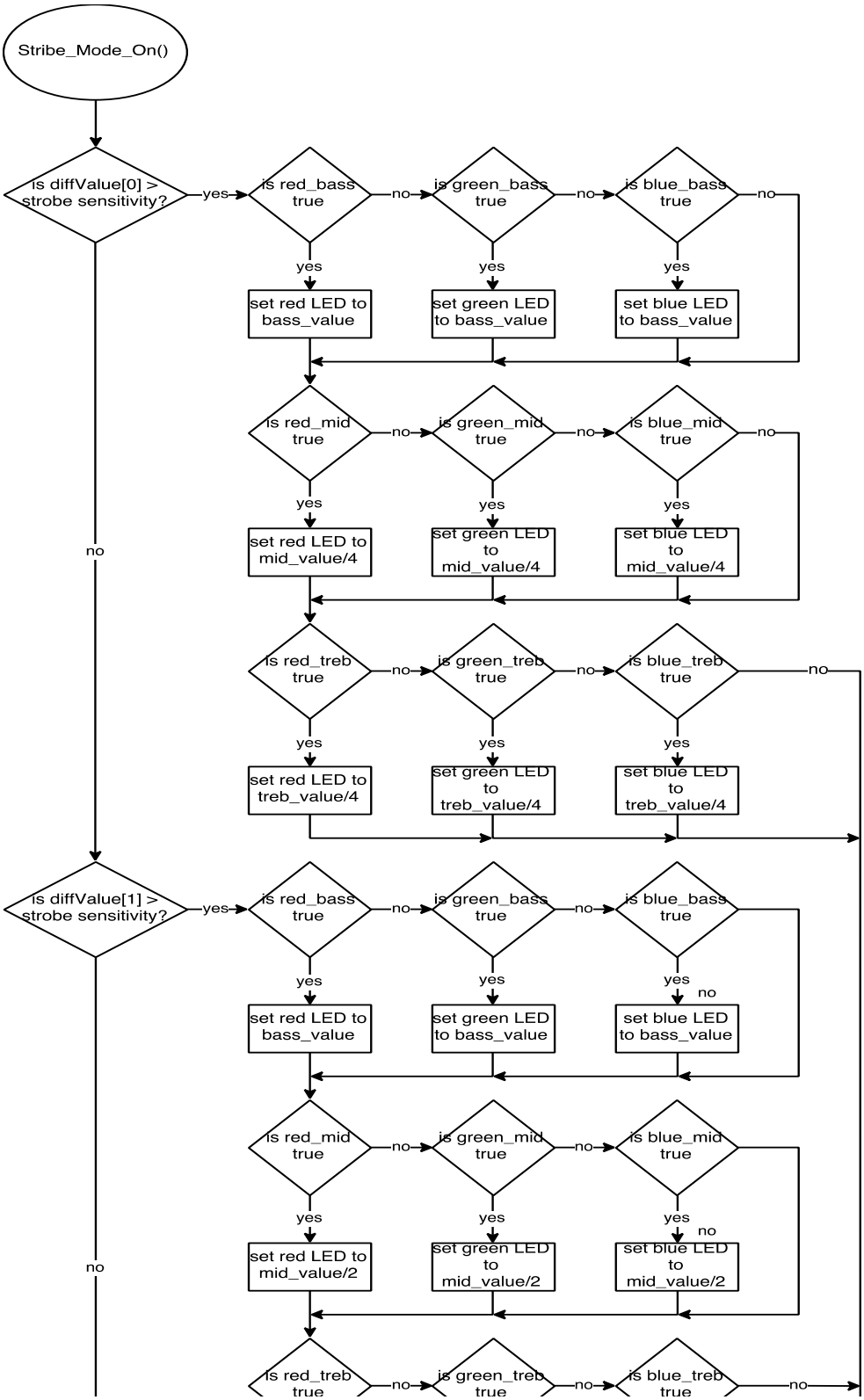

Zero_Mode()
is initial zero mode true?
yes
bass_value = 255
mid_value = 0
treb_value = 0
zero_mode_run = 0
intial_zero_mode = false
no
is zero_mode_run = 0?
yes
is bass_value > 0?
no
zero_mode_run = 1
yes
bass_value = bass_value - zero_mode_step
mid_value = mid_value + zero_mode_step
no
is zero_mode_run = 1?
yes
is mid_value > 0?
no
zero_mode_run = 2
yes
mid_value = mid_value - zero_mode_step
treb_value = treb_value + zero_mode_step
no
is zero_mode_run = 2?
yes
is treb_value > 0?
no
zero_mode_run = 0
yes
treb_value = treb_value - zero_mode_step
bass_value = bass_value + zero_mode_step
no
Strobe_Mode_Off()
delay zero_mode_delay (ms)
END

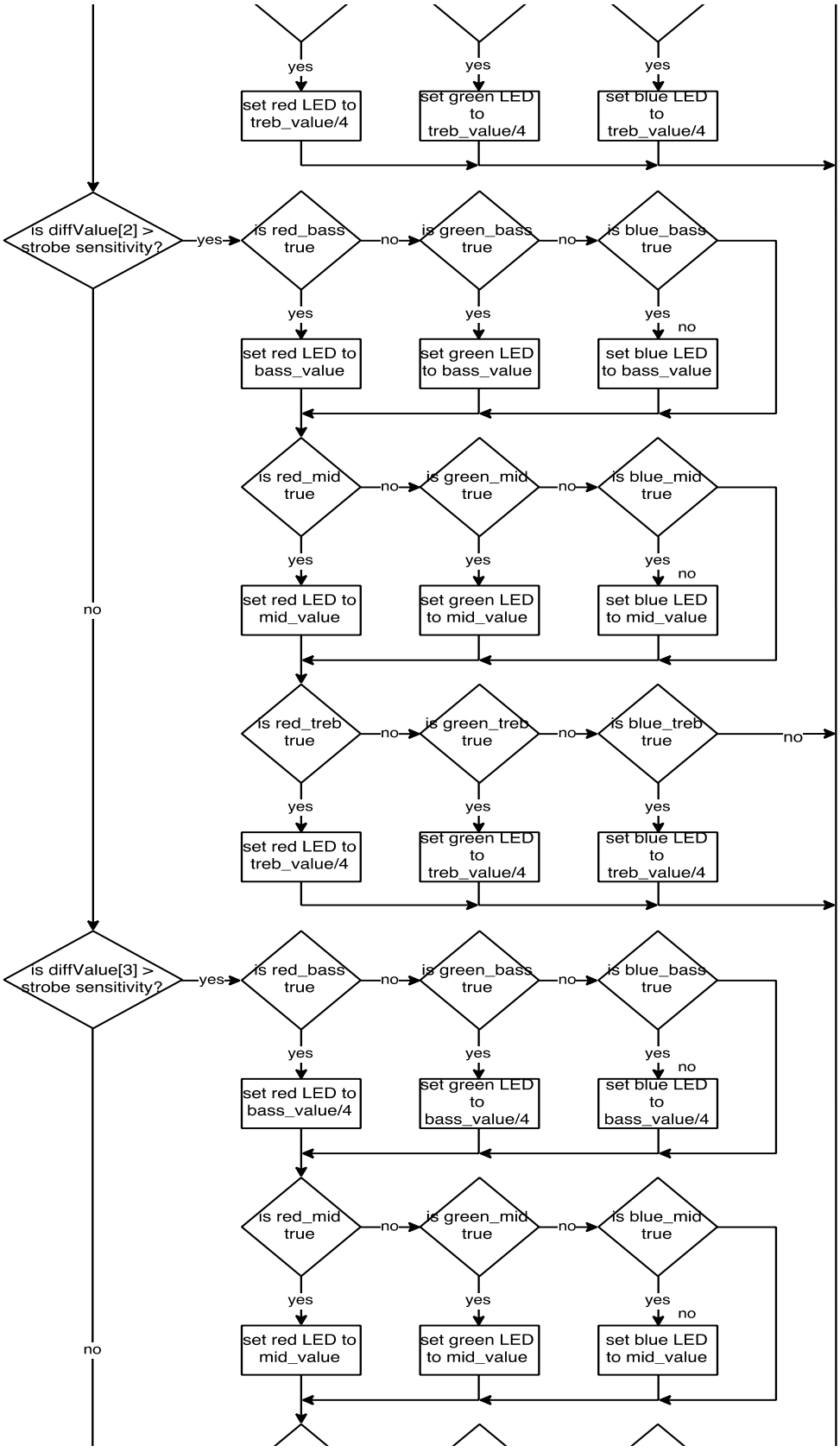

Define_Light_Min_and_Max()
is bass > mid > treb?
yes
bass_max = 255
mid_max = 200
treb_max = 150
no
is bass > treb > mid?
yes
bass_max = 255
mid_max = 150
treb_max = 200
no
is mid > bass > treb?
yes
bass_max = 200
mid_max = 255
treb_max = 150
no
is mid > treb > bass?
yes
bass_max = 150
mid_max = 255
treb_max = 200
no
is treb > bass > mid?
yes
bass_max = 200
mid_max = 150
treb_max = 255
no
is treb > mid > bass?
yes
bass_max = 150
mid_max = 200
treb_max = 255
no
is bass reading < bass_max?
yes
bass_min = bass reading
no
bass_min = bass_max
is mid reading < mid_max?
yes
mid_min = mid reading
no
mid_min = mid_max
is treble reading < treb_max?
yes
treb_min = treble reading
no
treb_min = treb_max
END

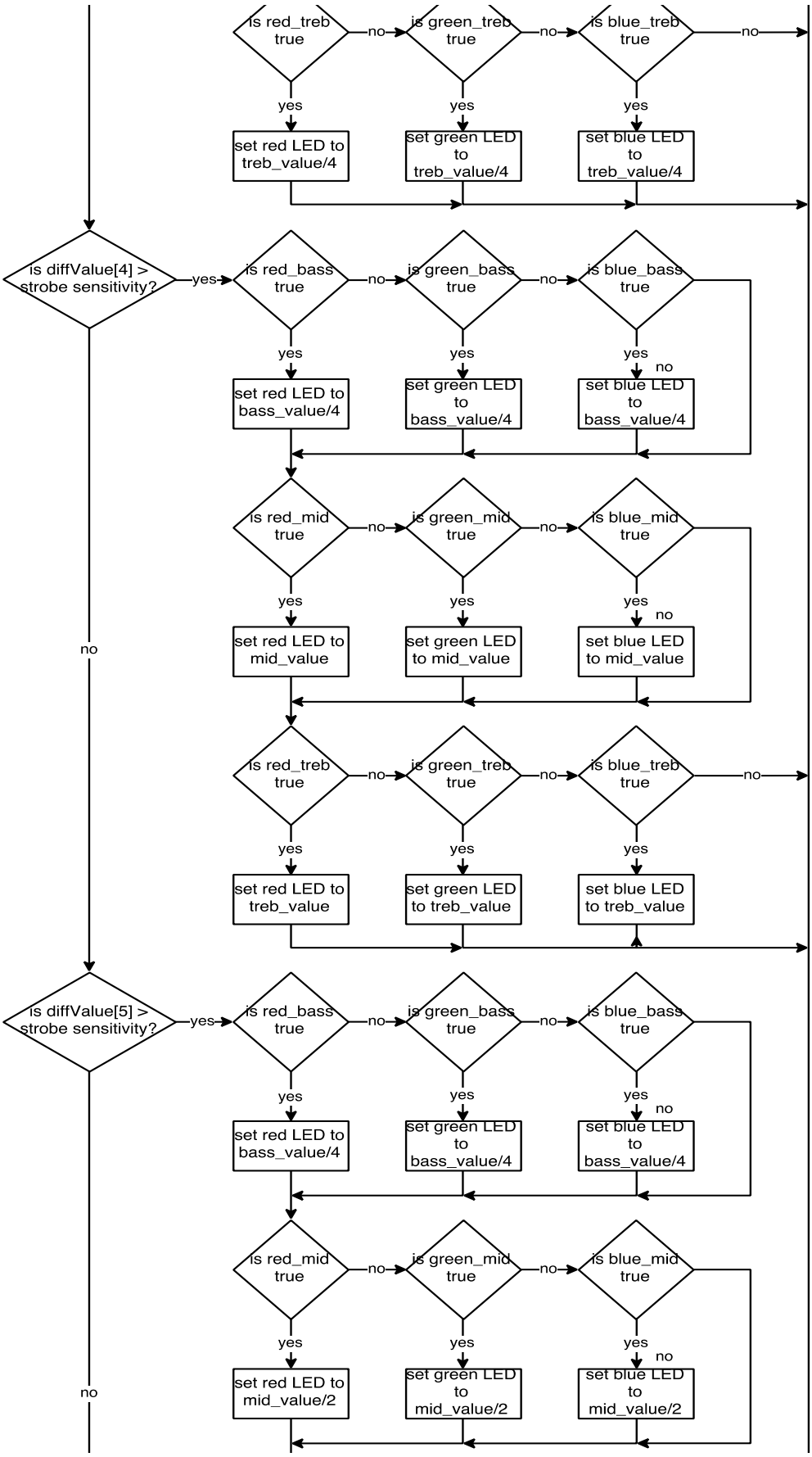

Define_Step_Size()
is bass reading > bass average + bass stdev
no
is bass reading > bass average
no
is bass reading < bass average
no
is bass reading < bass average - bass stdev
yes
yes
yes
yes
set bass_step = 2*stepsizeup
set bass_step = stepsizeup
set bass_step = stepsizedown
set bass_step = 2*stepsizedown
no
is mid reading > mid average + mid stdev
no
is mid reading > mid average
no
is mid reading < mid average
no
is mid reading < mid average - mid stdev
yes
yes
yes
yes
set mid_step = 2*stepsizeup
set mid_step = stepsizeup
set mid_step = stepsizedown
set mid_step = 2*stepsizedown
no
is treble reading > treble average + treble stdev
no
is treble reading > treble average
no
is treble reading < treble average
no
is treble reading < treble average - treble stdev
yes
yes
yes
yes
set treb_step = 2*stepsizeup
set treb_step = stepsizeup
set treb_step = stepsizedown
set treb_step = 2*stepsizedown
no
END

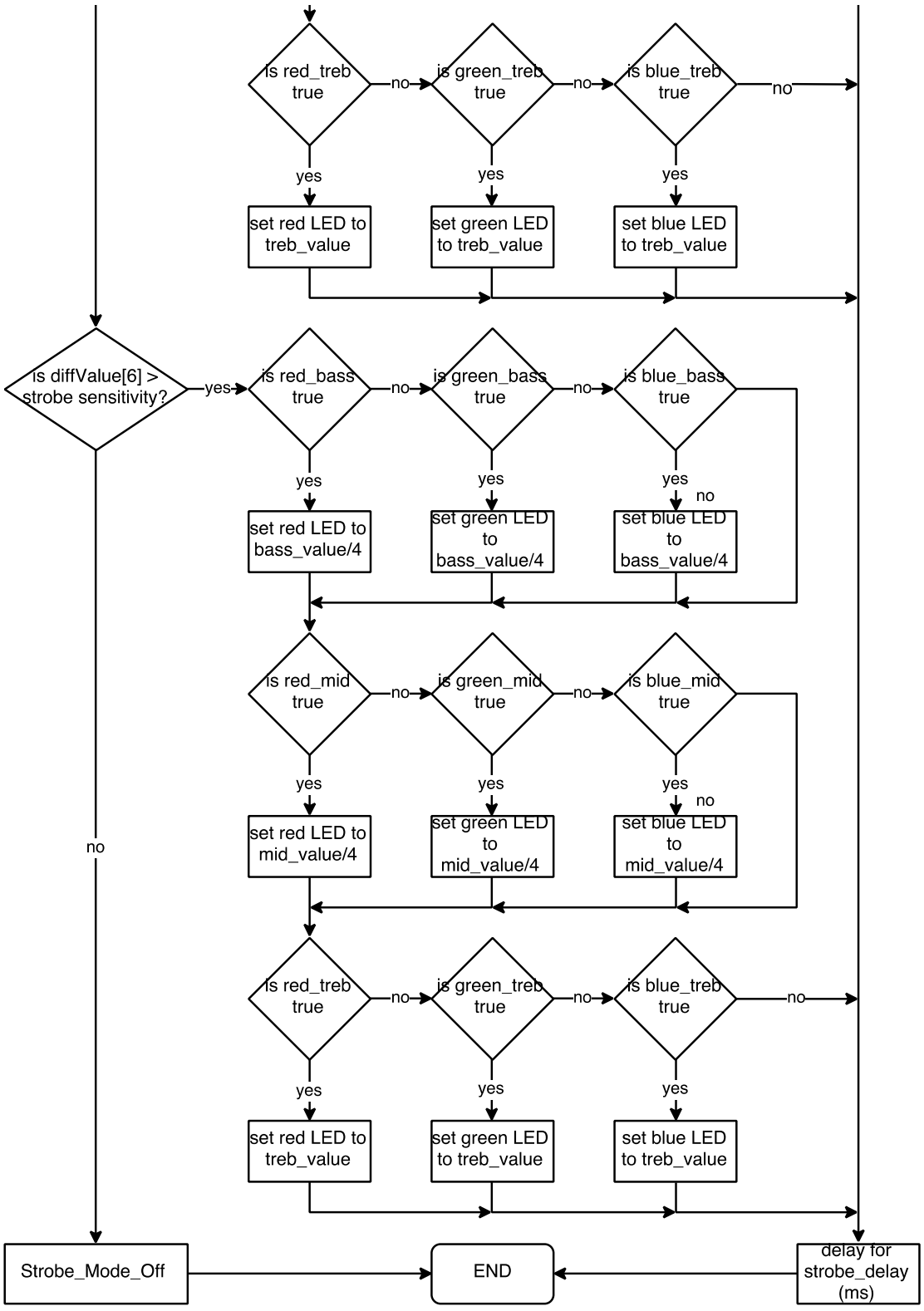

Define_Light_Values()
is bass value + bass step > bass max?
no
is bass value + bass step > bass min?
yes
yes
no
bass value = bass max
bass value = bass max
bass value = bass value + bass step
is mid value + mid step > mid max?
no
is mid value + mid step > mid min?
yes
yes
no
mid value = mid max
mid value = mid max
mid value = mid value + mid step
is treble value + treble step > treble max?
no
is treble value + treble step > treble min?
yes
yes
no
treble value = treble max
treble value = treble max
treble value = treble value + treble step
END


Strobe_Mode_Off()
is red_bass true
no
is green_bass true
no
is blue_bass true
no
yes
yes
yes
set red LED to bass_value
set green LED to bass_value
set blue LED to bass_value
is red_mid true
no
is green_mid true
no
is blue_mid true
no
yes
yes
yes
set red LED to mid_value
set green LED to mid_value
set blue LED to mid_value
is red_treb true
no
is green_treb true
no
is blue_treb true
no
yes
yes
yes
set red LED to treb_value
set green LED to treb_value
set blue LED to treb_value
END


Stribe_Mode_On()
is diffValue[0] > strobe sensitivity?
yes
is red_bass true
no
is green_bass true
no
is blue_bass true
no
yes
yes
yes
set red LED to bass_value
set green LED to bass_value
set blue LED to bass_value
is red_mid true
no
is green_mid true
no
is blue_mid true
no
yes
yes
yes
set red LED to mid_value/4
set green LED to mid_value/4
set blue LED to mid_value/4
no
is red_treb true
no
is green_treb true
no
is blue_treb true
no
yes
yes
yes
set red LED to treb_value/4
set green LED to treb_value/4
set blue LED to treb_value/4
is diffValue[1] > strobe sensitivity?
yes
is red_bass true
no
is green_bass true
no
is blue_bass true
yes
yes
yes
no
set red LED to bass_value
set green LED to bass_value
set blue LED to bass_value
is red_mid true
no
is green_mid true
no
is blue_mid true
yes
yes
yes
no
set red LED to mid_value/2
set green LED to mid_value/2
set blue LED to mid_value/2
no
is red_treb true
no
is green_treb true
no
is blue_treb true
no


yes
yes
yes
set red LED to treb_value/4
set green LED to treb_value/4
set blue LED to treb_value/4
is diffValue[2] > strobe sensitivity?
yes
is red_bass true
no
is green_bass true
no
is blue_bass true
yes
yes
yes
no
set red LED to bass_value
set green LED to bass_value
set blue LED to bass_value
is red_mid true
no
is green_mid true
no
is blue_mid true
yes
yes
yes
no
set red LED to mid_value
set green LED to mid_value
set blue LED to mid_value
no
is red_treb true
no
is green_treb true
no
is blue_treb true
no
yes
yes
yes
set red LED to treb_value/4
set green LED to treb_value/4
set blue LED to treb_value/4
is diffValue[3] > strobe sensitivity?
yes
is red_bass true
no
is green_bass true
no
is blue_bass true
yes
yes
yes
no
set red LED to bass_value/4
set green LED to bass_value/4
set blue LED to bass_value/4
is red_mid true
no
is green_mid true
no
is blue_mid true
yes
yes
yes
no
set red LED to mid_value
set green LED to mid_value
set blue LED to mid_value
no
no


is red_treb true
no
is green_treb true
no
is blue_treb true
no
yes
yes
yes
set red LED to treb_value/4
set green LED to treb_value/4
set blue LED to treb_value/4
is diffValue[4] > strobe sensitivity?
yes
is red_bass true
no
is green_bass true
no
is blue_bass true
yes
yes
yes
no
set red LED to bass_value/4
set green LED to bass_value/4
set blue LED to bass_value/4
is red_mid true
no
is green_mid true
no
is blue_mid true
yes
yes
yes
no
set red LED to mid_value
set green LED to mid_value
set blue LED to mid_value
no
is red_treb true
no
is green_treb true
no
is blue_treb true
no
yes
yes
yes
set red LED to treb_value
set green LED to treb_value
set blue LED to treb_value
is diffValue[5] > strobe sensitivity?
yes
is red_bass true
no
is green_bass true
no
is blue_bass true
yes
yes
yes
no
set red LED to bass_value/4
set green LED to bass_value/4
set blue LED to bass_value/4
is red_mid true
no
is green_mid true
no
is blue_mid true
yes
yes
yes
no
set red LED to mid_value/2
set green LED to mid_value/2
set blue LED to mid_value/2
no


is red_treb true
no
is green_treb true
no
is blue_treb true
no
yes
yes
yes
set red LED to treb_value
set green LED to treb_value
set blue LED to treb_value
is diffValue[6] > strobe sensitivity?
yes
is red_bass true
no
is green_bass true
no
is blue_bass true
yes
yes
yes
no
set red LED to bass_value/4
set green LED to bass_value/4
set blue LED to bass_value/4
is red_mid true
no
is green_mid true
no
is blue_mid true
yes
yes
yes
no
set red LED to mid_value/4
set green LED to mid_value/4
set blue LED to mid_value/4
no
is red_treb true
no
is green_treb true
no
is blue_treb true
no
yes
yes
yes
set red LED to treb_value
set green LED to treb_value
set blue LED to treb_value
Strobe_Mode_Off
END
delay for strobe_delay (ms)

Appendix A-4: Arduino Code

```
//* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
//* LED Lightshow for Mechatronics with Dr. Kevin Nikles                  *
//* By: Robert Hure and Josh King                                         *
//* This code takes the input from an audio signal processed by an MSGEQ7   *
//* and creates an RGB light show based off of the different frequencies    *
//* in the given audio signal.  The color of each frequency and the option  *
//* to turn on and off a strobe mode can be manipulated using a controlbox  *
//* with a three position switch, a button, and a potentiometer.  If no     *
//* music is detetected, the program will run a fade sequence.  This can    *
//* also be activated at anytime using the controlbox.                      *
//*                                                                         *
//* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
//_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ //

// Set up LED libary:
#include <LED.h>
LED red = LED(10);         // Define initial bass LED pin (D10)*
LED green = LED(3);           // Define initial midband LED pin (D3)*
LED blue = LED(11);        // Define initial treble LED pin (D11)*
//*Note: these must match the intial state in Color_Mode() and all be PWM pins
// This libary can be found at: http://playground.arduino.cc/Code/LED

// Set up Statistic libary:
#include <Statistic.h>
Statistic band1;      // Define band1 as a dataset
Statistic band3;      // Define band3 as a dataset
Statistic band5;      // Define band5 as a dataset
//This libary can be found at: http://playground.arduino.cc/Main/Statistics

// Define analog pins:
#define EQin 0             // MSGEQ7 OUT pin
#define strobe_pot 5       // Analog pin for the strobe sensitivity potentiomiter

// Define digital pins:
#define resetPin 2          // MSGEQ7 RESET pin
#define GREENPIN 3          // PWM pin connected to the green LED
#define strobePin 4         // MSGEQ7 STROBE pin
#define strobe_on_pin 6     // Digital pin for on state of the 3 position switch
#define strobe_off_pin 7    // Digital pin for the off state of the 3 position switch
#define color_switch 8      // Digital pin for button that runs through all possible pairings for the bands and colors
#define REDPIN 10           // PWM pin connected to the red LED
#define BLUEPIN 11          // PWM pin connected to the blue LED

// Global Variables:
int spectrumValue1[7];          // Last readings from MSGEQ7
int spectrumValue2[7];          // Current readings from MSGEQ7
int diffValue[7];               // Difference bewteen the two readings
int diff_bass = 0;              // Stored value for the difference bewteen last two bass bands (band1)
int diff_mid = 0;               // Stored value for the difference bewteen last two middle bands (band3)
int diff_treb = 0;              // Stored value for the difference bewteen last two treble bands (band5)
int bass_step = 0;              // Stored value for the change in brightness for the bass LED
int mid_step = 0;               // Stored value for the change in brightness for the midband LED
int treb_step = 0;              // Stored value for the change in brightness for the treble LED
```

```
int red_value = 0;              // Stored value for the brightness of red LED (0-255)
int green_value = 0;            // Stored value for the brightness of green LED (0-255)
int blue_value = 0;             // Stored value for the brightness of blue LED (0-255)
int bass_value = 0;             // Stored value for the brightness of the bass LED (0-255)
int mid_value = 0;              // Stored value for the brightness of the midband LED (0-255)
int treb_value = 0;             // Stored value for the brightness of the treble LED (0-255)
int bass_min = 0;               // Stored value for the minmum allowed bass light brightness
int mid_min = 0;                // Stored value for the minmum allowed midband light brightness
int treb_min = 0;               // Stored value for the minmum allowed treble light brightness
int bass_max = 255;             // Stored value for the maximum allowed bass light brightness
int mid_max = 255;              // Stored value for the maximum allowed midband light brightness
int treb_max = 255;             // Stored value for the maximum allowed treble light brightness
int color_mode = 0;             // Defines which LED color the bass, midband, and treble (0-5)
int zerocount = 0;              // Stored value for the number of times all bands read 0
int nonzerocount = 0;           // Stored value for the number of times not all bands read 0
int strobe_sensitivity = 0;     // Defines the maximum differnece required in a band for it to strobe
int button_state = 0;           // Stored value for current state of the button
int last_button_state = 0;      // Stored value for the previouse state of the button
int strobe_on = 0;              // Stored value for state of strobe on input from 3 position switch
int strobe_off = 0;             // Stored value for state of strobe off input from 3 position switch
int zero_mode_run = 0;          // Stored value for the state of fade sequince for zero mode

// Constants:                   (alter these to tune the code)
const int filterValue = 60;     // Value to remove noise
const int mapValue = 255;       // Value to map spectrumValues to
const int readnum = 20;         // Defines how many datapoints will be held for each band reading statistic
const int stepsizeup = 3;       // Alters the rate at which the LEDs will brighten
const int stepsizedown = -4;    // Alters the rate at which the LEDs will dim
const int zerocountmax = 80;    // Defines the number of counts needed to enter the zero state
const int nonzerocountmax = 5;  // Defines the number of counts needed to leave the zero state
const int zero_mode_step = 5;   // Defines the stepsize of each run for zero mode
const int zero_mode_delay = 50; // Defines delay between steps for zero mode (ms)
const int strobe_delay = 20;    // Defines length of strobe in strobe mode (ms)

// Boolean logic variables:
bool strobe_mode;               // True: strobe mode on                    False: strobe mode off
bool kill_switch;               // True: disable music response            False: enable music response
bool new_color_mode;            // True: redefine colors for each band     False: keep colors for each band
bool initial_zero_mode;         // True: reset values to intial state of zero mode    False: continue running zero mode
bool red_bass;                  // True: red is bass band                  False: red is not bass band
bool red_mid;                   // True: red is mid band                   False: red is not mid band
bool red_treb;                  // True: red is treble band                False: red is not treble band
bool green_bass;                // True: green is bass band                False: green is not bass band
bool green_mid;                 // True: green is mid band                 False: green is not mid band
bool green_treb;                // True: green is treble band              False: green is not treble band
bool blue_bass;                 // True: blue is bass band                 False: blue is not bass band
bool blue_mid;                  // True: blue is mid band                  False: blue is not mid band
bool blue_treb;                 // True: blue is treble band               False: blue is not treble band

void setup() {
  analogReference(DEFAULT);     // Set reference voltage: 5V
  Serial.begin(9600);           // Step up Serial in 9600 baud
  pinMode(EQin, INPUT);         // Set Input pins
  pinMode(strobe_on_pin, INPUT);
  pinMode(strobe_off_pin, INPUT);
  pinMode(strobe_pot, INPUT);
```

```
  pinMode(color_switch, INPUT);
  pinMode(strobePin, OUTPUT);        // Set output pins
  pinMode(resetPin, OUTPUT);
  digitalWrite(resetPin, LOW);       // Set startup values for putput pins
  digitalWrite(strobePin, HIGH);
  digitalWrite(resetPin, HIGH);      // Reset MSGEQ7
  digitalWrite(resetPin, LOW);
  band1.clear();                  // Clear statstics bands
  band3.clear();
  band5.clear();
  red_bass = true;
  green_mid = true;
  blue_treb = true;
  initial_zero_mode = true;
}

// Main loop:
void loop() {
  Check_Stats();                              // Resets all stats if they have more than readnum data points
  Read_Control_Box();                           // Reads and interperates all values from the control box
  Check_Color_Mode();                           // Changes to the next color layout if prompted
  Read_MSGEQ7();                              // Reads values form the MSGEQ7
  Check_Zeros();                             // Sets zerocount
  if (zerocount > zerocountmax || kill_switch == 1) {      // Conditions for zeromode
    Zero_Mode();                              // Sends a fade sequence
    Serial.print("Zeromode");
  }
  else {
  Define_Light_Min_and_Max();                       // Defines min and max values for each LED
  Define_Step_Size();                           // Defines change in brightness for each LED
  Define_Light_Values();                          // Defines the brightness for each LED
  if (strobe_mode == false) {                       // Conditions for strobe mode off
    Strobe_Mode_Off();                            // Sets light values without strobing
      initial_zero_mode = true;
    Serial.print("StrobeOff");
  }
  else if (strobe_mode == true) {                     // Conditions for strobe mode on
    Strobe_Mode_On();                             // Sets LED values and stobes based off of the differnce spectrum
    initial_zero_mode = true;
    Serial.print("StrobeOn");
  }
  }
  Serial.println();
}

int Check_Stats() {
  if (band1.count() == readnum) {        // Clears band1 if it reaches the max number of datapoints
    band1.clear();
    band1.add(spectrumValue1[1]);        // Adds the last two readings to the next dataset
    band1.add(spectrumValue2[1]);
  }
  if (band3.count() == readnum) {        // Clears band3 if it reaches the max number of datapoints
    band3.clear();
    band3.add(spectrumValue1[3]);        // Adds the last two readings to the next dataset
    band3.add(spectrumValue2[3]);
  }
```

```
  if (band5.count() == readnum) {         // Clears band5 if it reaches the max number of datapoints
    band5.clear();
    band5.add(spectrumValue1[5]);         // Adds the last two readings to the next dataset
    band5.add(spectrumValue2[5]);
  }
}

int Read_MSGEQ7() {
  digitalWrite(resetPin, HIGH);                                          // Reset MSGEQ7
  digitalWrite(resetPin, LOW);
  for (int i = 0; i < 7; i++) {
    spectrumValue1[i] = spectrumValue2[i];                                // Store previous reading in spectrum1
    digitalWrite(strobePin, LOW);                                        // Tell MSGEQ7 to send band
    delayMicroseconds(20);                                               // Allow output to settle
    spectrumValue2[i] = analogRead(EQin);                                 // ADC on MSGEQ7 output
    spectrumValue2[i] = constrain(spectrumValue2[i], filterValue, 1023);          // Constrain any value above
1023 or below filterValue
    spectrumValue2[i] = map(spectrumValue2[i], filterValue, 1023, 0, mapValue);       // Remap the value to a
number between 0 and mapValue
    diffValue[i] = spectrumValue2[i] - spectrumValue1[i];                  // Calculate difference value
    digitalWrite(strobePin, HIGH);                                       // Set strobe on the MSGEQ7 to default state
    Serial.print(spectrumValue2[i]);                                     // Send results to serial monitor
    Serial.print(", ");
  }
  band1.add(spectrumValue2[1]);                                          // Add spectrum values into respective data
sets
  band3.add(spectrumValue2[3]);
  band5.add(spectrumValue2[5]);
}

int Check_Color_Mode() {
  if (new_color_mode == true) {                    // Only procede if new_color_mode is true
    if (color_mode == 0) {                         // Sets up boolean logic for relation between color and frequency
band
      red_bass = true;   red_mid = false;   red_treb = false;
      green_bass = false; green_mid = true; green_treb = false;
      blue_bass = false;  blue_mid = false;  blue_treb = true;
      Serial.print("colormode: 0");                  // Prints state to serial when the color mode changes
    }
    else if (color_mode == 1) {
      red_bass = true;    red_mid = false;   red_treb = false;
      green_bass = false; green_mid = false; green_treb = true;
      blue_bass = false;  blue_mid = true;   blue_treb = false;
      Serial.print("colormode: 1");
    }
    else if (color_mode == 2) {
      red_bass = false;   red_mid = true;    red_treb = false;
      green_bass = true;  green_mid = false; green_treb = false;
      blue_bass = false;  blue_mid = false;  blue_treb = true;
      Serial.print("colormode: 2");
    }
    else if (color_mode == 3) {
      red_bass = false;   red_mid = true;    red_treb = false;
      green_bass = false; green_mid = false; green_treb = true;
      blue_bass = true;   blue_mid = false;  blue_treb = false;
      Serial.print("colormode: 3");
```

```
    }
    else if (color_mode == 4) {
      red_bass = false;   red_mid = false;   red_treb = true;
      green_bass = true;  green_mid = false; green_treb = false;
      blue_bass = false;  blue_mid = true;   blue_treb = false;
      Serial.print("colormode: 4");
    }
    else if (color_mode == 5) {
      red_bass = false;   red_mid = false;   red_treb = true;
      green_bass = false; green_mid = true;  green_treb = false;
      blue_bass = true;   blue_mid = false;  blue_treb = false;
      Serial.print("colormode: 5");
    }
    new_color_mode = false;                         // Set new_color_mode to flase to prevent code from setting up
the LED pins each run
    delay(10);
  }
}

int Define_Light_Min_and_Max() {                                                    // Sets maximum values to each
band
    if (spectrumValue2[1] > spectrumValue2[3] && spectrumValue2[3] > spectrumValue2[5]) {       // when strobe
is off to prevent solid
      bass_max = 255;                                                   // white light due to clipping
      mid_max = 200;
      treb_max = 150;
    }
    else if (spectrumValue2[1] > spectrumValue2[5] && spectrumValue2[5] > spectrumValue2[3]) {
      bass_max = 255;
      mid_max = 150;
      treb_max = 200;
    }
    else if (spectrumValue2[3] > spectrumValue2[1] && spectrumValue2[1] > spectrumValue2[5]) {
      bass_max = 200;
      mid_max = 255;
      treb_max = 150;
    }
    else if (spectrumValue2[3] > spectrumValue2[5] && spectrumValue2[5] > spectrumValue2[1]) {
      bass_max = 150;
      mid_max = 255;
      treb_max = 200;
    }
    else if (spectrumValue2[5] > spectrumValue2[1] && spectrumValue2[1] > spectrumValue2[3]) {
      bass_max = 200;
      mid_max = 150;
      treb_max = 255;
    }
    else if (spectrumValue2[5] > spectrumValue2[3] && spectrumValue2[3] > spectrumValue2[1]) {
      bass_max = 150;
      mid_max = 200;
      treb_max = 255;
    }
  else {                                    // When strobe mode is on all bands have highest value as maximum
    bass_max = 255;
    mid_max = 255;
    treb_max = 255;
```

```
  }
  if (spectrumValue2[1] < bass_max) {                  // Sets the minimum value to the current reading if it is below
the max
    bass_min = spectrumValue2[1];
  }
  else {
    bass_min = bass_max;                               // Sets the minimum equal to the maximum otherwise
  }
  if (spectrumValue2[3] < mid_max) {
    mid_min = spectrumValue2[3];
  }
  else {
    mid_min = mid_max;
  }
  if (spectrumValue2[5] < treb_max) {
    treb_min = spectrumValue2[5];
  }
  else {
    treb_min = treb_max;
  }
}

int Define_Step_Size() {
  if (spectrumValue2[1] > band1.average() + band1.pop_stdev()) {          // If the current reading is more than 1
standard deviation above the average
    bass_step = 2 * stepsizeup;                                  // set the bass step to twice the up step size
  }
  else if (spectrumValue2[1] > band1.average()) {                  // Otherwise if the current reading is above the
average
    bass_step = stepsizeup;                                  // set the bass step to the up step size
  }
  else if (spectrumValue2[1] < band1.average()) {                  // Otherwise if the current reading is below the
average
    bass_step = stepsizedown;                                  // set the bass step to the step size down
  }
  else if (spectrumValue2[1] < band1.average() - band1.pop_stdev()) {        // Otherwise if the current reading is
more than 1 standard deviation below the average
    bass_step = 2 * stepsizedown;                                  // set the bass step to twice the step size down
  }
  if (spectrumValue2[3] > band3.average() + band3.pop_stdev()) {          // If the current reading is more than 1
standard deviation above the average
    mid_step = 2 * stepsizeup;                                  // set the mid step to twice the up step size
  }
  else if (spectrumValue2[3] > band3.average()) {                  // Otherwise if the current reading is above the
average
    mid_step = stepsizeup;                                  // set the mid step to the up step size
  }
  else if (spectrumValue2[3] < band3.average()) {                  // Otherwise if the current reading is below the
average
    mid_step = stepsizedown;                                  // set the mid step to the step size down
  }
  else if (spectrumValue2[3] < band3.average() - band3.pop_stdev()) {        // Otherwise if the current reading is
more than 1 standard deviation below the average
    mid_step = 2 * stepsizedown;                                  // set the mid step to twice the step size down
  }
```

```
  if (spectrumValue2[5] > band5.average() + band5.pop_stdev()) {            // If the current reading is more than 1
standard deviation above the average
    treb_step = 2 * stepsizeup;                                    // set the treble step to twice the up step size
  }
  else if (spectrumValue2[5] > band5.average())  {                 // Otherwise if the current reading is above the
average
    treb_step = stepsizeup;                                        // set the treble step to the up step size
  }
  else if (spectrumValue2[5] < band5.average()) {                  // Otherwise if the current reading is below the
average
    treb_step = stepsizedown;                                      // set the trebl step to the step size down
  }
  else if (spectrumValue2[5] < band5.average() - band5.pop_stdev()) {        // Otherwise if the current reading is
more than 1 standard deviation below the average
    treb_step = 2 * stepsizedown;                                  // set the treble step to twice the step size down
  }
}

int Define_Light_Values() {
  if (bass_value + bass_step > bass_max) {         // Prevents the bass value from going above the max
    bass_value = bass_max;
  }
  else if (bass_value + bass_step < bass_min) {    // Prevents the bass value from going below the min
    bass_value = bass_min;
  }
  else {                                           // Otherwise changes bass value by bass step
    bass_value = bass_value + bass_step;
  }
  if (mid_value + mid_step > mid_max) {            // Prevents the midband value from going above the max
    mid_value = mid_max;
  }
  else if (mid_value + mid_step < mid_min) {       // Prevents the midband value from going below the min
    mid_value = mid_min;
  }
  else {                                           // Otherwise changes midband value by mid step
    mid_value = mid_value + mid_step;
  }
  if (treb_value + treb_step > treb_max) {         // Prevents the treble value from going above the max
    treb_value = treb_max;
  }
  else if (treb_value + treb_step < treb_min) {    // Prevents the treble value from going below the min
    treb_value = treb_min;
  }
  else {                                           // Otherwise changes treble value by treb step
    treb_value = treb_value + treb_step;
  }
}

int Check_Zeros() {
  if (spectrumValue2[0] == 0 && spectrumValue2[1] == 0 && spectrumValue2[2] == 0                    // If
all bands are 0 increase zero count
      && spectrumValue2[3] == 0 && spectrumValue2[4] == 0 && spectrumValue2[5] == 0 && spectrumValue2[6]
== 0) {
    nonzerocount = 0;
    zerocount ++;
  }
```

```
  else {                                                    // Otherwise set zero count to 0
    nonzerocount ++;
    if (nonzerocount > nonzerocountmax){
    zerocount = 0;
    }
  }
}

int Strobe_Mode_On() {
  if (diffValue[0] > strobe_sensitivity) {          // If band 0 has a difference reading larger than strobe_sensitivity
    if (red_bass == true){                      // If red is bass band
      red.setValue(bass_value);                   // Set red LED to bass value
    }
    else if (green_bass == true){                // If green is bass band
      green.setValue(bass_value);                 // Set green LED to bass value
    }
    else if (blue_bass == true){                // If blue is bass band
      blue.setValue(bass_value);                  // Set blue LEd to bass value
    }
    if (red_mid == true){                       // If red is mid band
      red.setValue(mid_value/4);                  // Set red LED to 1/4 mid value
    }
    else if (green_mid == true){                // If green is mid band
     green.setValue(mid_value/4);                 // Set green LED to 1/4 mid value
    }
    else if (blue_mid == true){                 // If blue is mid band
     blue.setValue(mid_value/4);                  // Set blue LED to 1/4 mid value
    }
    if (red_treb == true){                      // If red is treble band
      red.setValue(treb_value/4);                 // Set red LED to 1/4 treble value
    }
    else if (green_treb == true){               // If green is treble band
      green.setValue(treb_value/4);               // Set green LED to 1/4 treble value
    }
    else if (blue_treb == true){                // If blue is treble band
      blue.setValue(treb_value/4);                // Set blue LED to 1/4 treble value
    }
    delay(strobe_delay);                        // Delay to hold values
  }
  else if (diffValue[1] > strobe_sensitivity) {       // If band 1 has a difference reading larger than strobe_sensitivity
    if (red_bass == true){                      // If red is bass band
      red.setValue(bass_value);                   // Set red LED to bass value
    }
    else if (green_bass == true){                // If green is bass band
      green.setValue(bass_value);                 // Set green LED to bass value
    }
    else if (blue_bass == true){                // If blue is bass band
      blue.setValue(bass_value);                  // Set blue LEd to bass value
    }
    if (red_mid == true){                       // If red is mid band
      red.setValue(mid_value/2);                  // Set red LED to 1/2 mid value
    }
    else if (green_mid == true){                // If green is mid band
     green.setValue(mid_value/2);                 // Set green LED to 1/2 mid value
    }
    else if (blue_mid == true){                 // If blue is mid band
```

```
   blue.setValue(mid_value/2);                  // Set blue LED to 1/2 mid value
  }
  if (red_treb == true){                       // If red is treble band
    red.setValue(treb_value/4);                 // Set red LED to 1/4 treble value
  }
  else if (green_treb == true){                // If green is treble band
    green.setValue(treb_value/4);               // Set green LED to 1/4 treble value
  }
  else if (blue_treb == true){                 // If blue is treble band
    blue.setValue(treb_value/4);                // Set blue LED to 1/4 treble value
  }
  delay(strobe_delay);                         // Delay to hold values
}
else if (diffValue[2] > strobe_sensitivity) {       // If band 2 has a difference reading larger than strobe_sensitivity
  if (red_bass == true){                       // If red is bass band
    red.setValue(bass_value);                   // Set red LED to bass value
  }
  else if (green_bass == true){                // If green is bass band
    green.setValue(bass_value);                 // Set green LED to bass value
  }
  else if (blue_bass == true){                 // If blue is bass band
    blue.setValue(bass_value);                  // Set blue LEd to bass value
  }
  if (red_mid == true){                        // If red is mid band
    red.setValue(mid_value);                    // Set red LED to mid value
  }
  else if (green_mid == true){                 // If green is mid band
   green.setValue(mid_value);                   // Set green LED to mid value
  }
  else if (blue_mid == true){                  // If blue is mid band
   blue.setValue(mid_value);                    // Set blue LED to mid value
  }
  if (red_treb == true){                       // If red is treble band
    red.setValue(treb_value/4);                 // Set red LED to 1/4 treble value
  }
  else if (green_treb == true){                // If green is treble band
    green.setValue(treb_value/4);               // Set green LED to 1/4 treble value
  }
  else if (blue_treb == true){                 // If blue is treble band
    blue.setValue(treb_value/4);                // Set blue LED to 1/4 treble value
  }
  delay(strobe_delay);                         // Delay to hold values
}
else if (diffValue[3] > strobe_sensitivity) {       // If band 3 has a difference reading larger than strobe_sensitivity
  if (red_bass == true){                       // If red is bass band
    red.setValue(bass_value/4);                 // Set red LED to 1/4 bass value
  }
  else if (green_bass == true){                // If green is bass band
    green.setValue(bass_value/4);               // Set green LED to 1/4 bass value
  }
  else if (blue_bass == true){                 // If blue is bass band
    blue.setValue(bass_value/4);                // Set blue LEd to 1/4 bass value
  }
  if (red_mid == true){                        // If red is mid band
    red.setValue(mid_value);                    // Set red LED to mid value
  }
```

```
  else if (green_mid == true){                  // If green is mid band
   green.setValue(mid_value);                    // Set green LED to mid value
  }
  else if (blue_mid == true){                   // If blue is mid band
   blue.setValue(mid_value);                     // Set blue LED to mid value
  }
  if (red_treb == true){                       // If red is treble band
    red.setValue(treb_value/4);                  // Set red LED to 1/4 treble value
  }
  else if (green_treb == true){                 // If green is treble band
    green.setValue(treb_value/4);                // Set green LED to 1/4 treble value
  }
  else if (blue_treb == true){                  // If blue is treble band
    blue.setValue(treb_value/4);                 // Set blue LED to 1/4 treble value
  }
  delay(strobe_delay);                         // Delay to hold values
}
else if (diffValue[4] > strobe_sensitivity) {       // If band 4 has a difference reading larger than strobe_sensitivity
  if (red_bass == true){                       // If red is bass band
    red.setValue(bass_value/4);                  // Set red LED to 1/4 bass value
  }
  else if (green_bass == true){                 // If green is bass band
    green.setValue(bass_value/4);                // Set green LED to 1/4 bass value
  }
  else if (blue_bass == true){                  // If blue is bass band
    blue.setValue(bass_value/4);                 // Set blue LEd to 1/4 bass value
  }
  if (red_mid == true){                        // If red is mid band
    red.setValue(mid_value);                     // Set red LED to mid value
  }
  else if (green_mid == true){                  // If green is mid band
   green.setValue(mid_value);                    // Set green LED to mid value
  }
  else if (blue_mid == true){                   // If blue is mid band
   blue.setValue(mid_value);                     // Set blue LED to mid value
  }
  if (red_treb == true){                       // If red is treble band
    red.setValue(treb_value);                    // Set red LED to treble value
  }
  else if (green_treb == true){                 // If green is treble band
    green.setValue(treb_value);                  // Set green LED to treble value
  }
  else if (blue_treb == true){                  // If blue is treble band
    blue.setValue(treb_value);                   // Set blue LED to treble value
  }
  delay(strobe_delay);                         // Delay to hold values
}
else if (diffValue[5] > strobe_sensitivity) {       // If band 5 has a difference reading larger than strobe_sensitivity
  if (red_bass == true){                       // If red is bass band
    red.setValue(bass_value/4);                  // Set red LED to 1/4 bass value
  }
  else if (green_bass == true){                 // If green is bass band
    green.setValue(bass_value/4);                // Set green LED to 1/4 bass value
  }
  else if (blue_bass == true){                  // If blue is bass band
    blue.setValue(bass_value/4);                 // Set blue LEd to 1/4 bass value
```

```
  }
  if (red_mid == true){                    // If red is mid band
   red.setValue(mid_value/2);                 // Set red LED to 1/2 mid value
  }
  else if (green_mid == true){               // If green is mid band
   green.setValue(mid_value/2);                // Set green LED to 1/2 mid value
  }
  else if (blue_mid == true){                // If blue is mid band
   blue.setValue(mid_value/2);                 // Set blue LED to 1/2 mid value
  }
  if (red_treb == true){                   // If red is treble band
   red.setValue(treb_value);                  // Set red LED to treble value
  }
  else if (green_treb == true){              // If green is treble band
   green.setValue(treb_value);                 // Set green LED to treble value
  }
  else if (blue_treb == true){               // If blue is treble band
   blue.setValue(treb_value);                  // Set blue LED to treble value
  }
  delay(strobe_delay);                     // Delay to hold values
 }
 else if (diffValue[6] > strobe_sensitivity) {       // If band 6 has a difference reading larger than strobe_sensitivity
  if (red_bass == true){                   // If red is bass band
   red.setValue(bass_value/4);                // Set red LED to 1/4 bass value
  }
  else if (green_bass == true){              // If green is bass band
   green.setValue(bass_value/4);               // Set green LED to 1/4 bass value
  }
  else if (blue_bass == true){               // If blue is bass band
   blue.setValue(bass_value/4);                // Set blue LEd to 1/4 bass value
  }
  if (red_mid == true){                    // If red is mid band
   red.setValue(mid_value/4);                 // Set red LED to 1/4 mid value
  }
  else if (green_mid == true){               // If green is mid band
   green.setValue(mid_value/4);                // Set green LED to 1/4 mid value
  }
  else if (blue_mid == true){                // If blue is mid band
   blue.setValue(mid_value/4);                 // Set blue LED to 1/4 mid value
  }
  if (red_treb == true){                   // If red is treble band
   red.setValue(treb_value);                  // Set red LED to treble value
  }
  else if (green_treb == true){              // If green is treble band
   green.setValue(treb_value);                 // Set green LED to treble value
  }
  else if (blue_treb == true){               // If blue is treble band
   blue.setValue(treb_value);                  // Set blue LED to treble value
  }
  delay(strobe_delay);                     // Delay to hold values
 }
 else {                                  // If no bands have a difference reading larger than strobe_sensitivity
  Strobe_Mode_Off();                         // Run strobe mode off
 }
}
```

```
int Strobe_Mode_Off() {
   if (red_bass == true){                    // If red is bass band
     red.setValue(bass_value);               // Set red LED to bass value
   }
   else if (green_bass == true){             // If green is bass band
     green.setValue(bass_value);             // Set green LED to bass value
   }
   else if (blue_bass == true){              // If blue is bass band
     blue.setValue(bass_value);              // Set blue LEd to bass value
   }
   if (red_mid == true){                     // If red is mid band
     red.setValue(mid_value);                // Set red LED to mid value
   }
   else if (green_mid == true){              // If green is mid band
    green.setValue(mid_value);               // Set green LED to mid value
   }
   else if (blue_mid == true){               // If blue is mid band
    blue.setValue(mid_value);                // Set blue LED to mid value
   }
   if (red_treb == true){                    // If red is treble band
     red.setValue(treb_value);               // Set red LED to treble value
   }
   else if (green_treb == true){             // If green is treble band
     green.setValue(treb_value);             // Set green LED to treble value
   }
   else if (blue_treb == true){              // If blue is treble band
     blue.setValue(treb_value);              // Set blue LED to treble value
   }
  }

int Zero_Mode() {
  if (initial_zero_mode == true) {                  // If told to reset
    bass_value = 255;                               // Set bass to max
    mid_value = 0;                                  // Set mid to 0
    treb_value = 0;                                 // Set treble to 0
    zero_mode_run = 0;                              // Reset the run number
    initial_zero_mode = false;                      // Done reseting
  }
  else {
    if (zero_mode_run == 0) {                       // For run 0: decrease bass and increase mid untill bass is 0
      if (bass_value > 0) {                         // and mid is 255 then advance run number
        bass_value = bass_value - zero_mode_step;
        mid_value = mid_value + zero_mode_step;
      }
      else {
        zero_mode_run = 1;
      }
    }
    if (zero_mode_run == 1) {                       // For run 1: decrease mid and increase treble untill mid is 0
      if (mid_value > 0) {                          // and treble is 255 then advance run number
        mid_value = mid_value - zero_mode_step;
        treb_value = treb_value + zero_mode_step;
      }
      else {
        zero_mode_run = 2;
      }
```

```
   }
   if (zero_mode_run == 2) {                    // For run 2: decrease treble and increase bass untill treble is 0
    if (treb_value > 0) {                       // and bass is 255 then advance run number
     bass_value = bass_value + zero_mode_step;
     treb_value = treb_value - zero_mode_step;
    }
    else {
     zero_mode_run = 0;
    }
   }
 }
 Strobe_Mode_Off();                             // Set light values
 delay(zero_mode_delay);                        // Delay to slow fade sequence
 Serial.print(initial_zero_mode);
 Serial.print(", ");
 Serial.print(zero_mode_run);
}
// Setting the values for the LEDs after each step instead of using a fade command
// allows the code to leave zero mode at any point in the fade sequance rather than
// at just the ends of a cycle.  This was an issue the first demo had trouble with

int Read_Control_Box() {
 button_state = digitalRead(color_switch);                  // Store the current state of the button
 if (button_state != last_button_state && button_state == LOW) {    // If the button if low and is not equal to the
previous state
   if (color_mode < 5) {
    color_mode ++;                                  // Increase color mode if less than 5
   }
   else {                                           // If color mode is 5
    color_mode = 0;                                 // Set color mode to 0
   }
   new_color_mode = true;                           // Tell code to switch color mode
   initial_zero_mode = true;                        // Tell code to reset zero mode
   delay(50);                                       // Delay to prevent bouncing
 }
 last_button_state = digitalRead(color_switch);             // Store last button state for next run
 strobe_on = digitalRead(strobe_on_pin);                    // Store strobe on pin reading
 strobe_off = digitalRead(strobe_off_pin);                  // Store strobe off pin reading
 if (strobe_on == 1) {                                      // If the strobe on pin is high
  kill_switch = false;                                      // Tell code not to go to zero mode
  strobe_mode = true;                                       // Tell code to go to strobe on mode
 }
 else if (strobe_off == 1) {                                // If the strobe off pin is high
  kill_switch = false;                                      // Tell code not to go to zero mode
  strobe_mode = false;                                      // Tell code to go to srobe off mode
 }
 else if (strobe_on == 0 && strobe_off == 0) {              // If both pins are low
  kill_switch = true;                                       // Tell code to go to zero mode
 }
 strobe_sensitivity = analogRead(strobe_pot);               // Read value from potentiomiter (ADC)
 strobe_sensitivity = map(strobe_sensitivity, 0, 1023, 0, 50);      // map value from 0 to 1023 to 0 to 50
}
```